Projektname:	Bankomponeten
Projektnummer:	0001
Kurzbeschreibung:	Erzeugung von Softwarekomponente(n) als dynamisch nachladbare Bibliotheken (DLLs) in C/C++, in der die grundlegenden Funktionen einer Bank abgebildet werden. Anschließend, Erstellung auf Basis der von einem anderen Team übernommenen DLLs und der eigenen DLLs ein gemeinsames C#/.NET Interface und das darauf basierend .NET Assemblies, die objektorientiert unter C# und .NET – dem eigenen DLL und dem übernommenen DLL als Wrapper/Adaptor/Bridge dienen. Es wird die volle bzw. geforderte Funktionalität der DLLs anbieten, ohne jedoch Details der DLLs und deren API kennen zu müssen.
Version:	1.2.4
Vorgelegt von:	Reimar Klammer, Johannes Probst, Daniel Komohorov
Author:	Daniel Komohorov
Auftraggeber:	DI (FH) DI Roland Graf, MSc
Erstelldatum:	22.11.2016

DOKUMENTENHISTORIE

Autor	Datum	Version	Änderung
Komohorov	26.10.2016	1.2.0	Erstellung der Dokumentation C/C++ DLL's (LB_01)
Klammer	27.10.2016	1.2.1	Ergänzung Erklärung zu Moduls und Namspaces (LB_02)
Klammer	30.10.2016	1.2.2	Ergänzung Erklärung zu Moduls und Namspaces (LB_02)
Komohorov	14.11.2016	1.2.3	Ergänzung: drei weiteren Komponenten (foreign Bank)
Komohorov	22.11.2016	1.2.4	Dok LB_01 in ein einzelnes Gesamtdokument eingebunden
Komohorov	24.11.2016	1.2.4	Offene Punkte angepasst

I Inhaltsverzeichnis

1 Dokumentation der Softwarekomponenten

1.1 Dokumentation der Bibliothekinterfaces, C/C++ Komponenten (LB_01)

1.1.1 Komponentenübersicht C/C++ DLL's

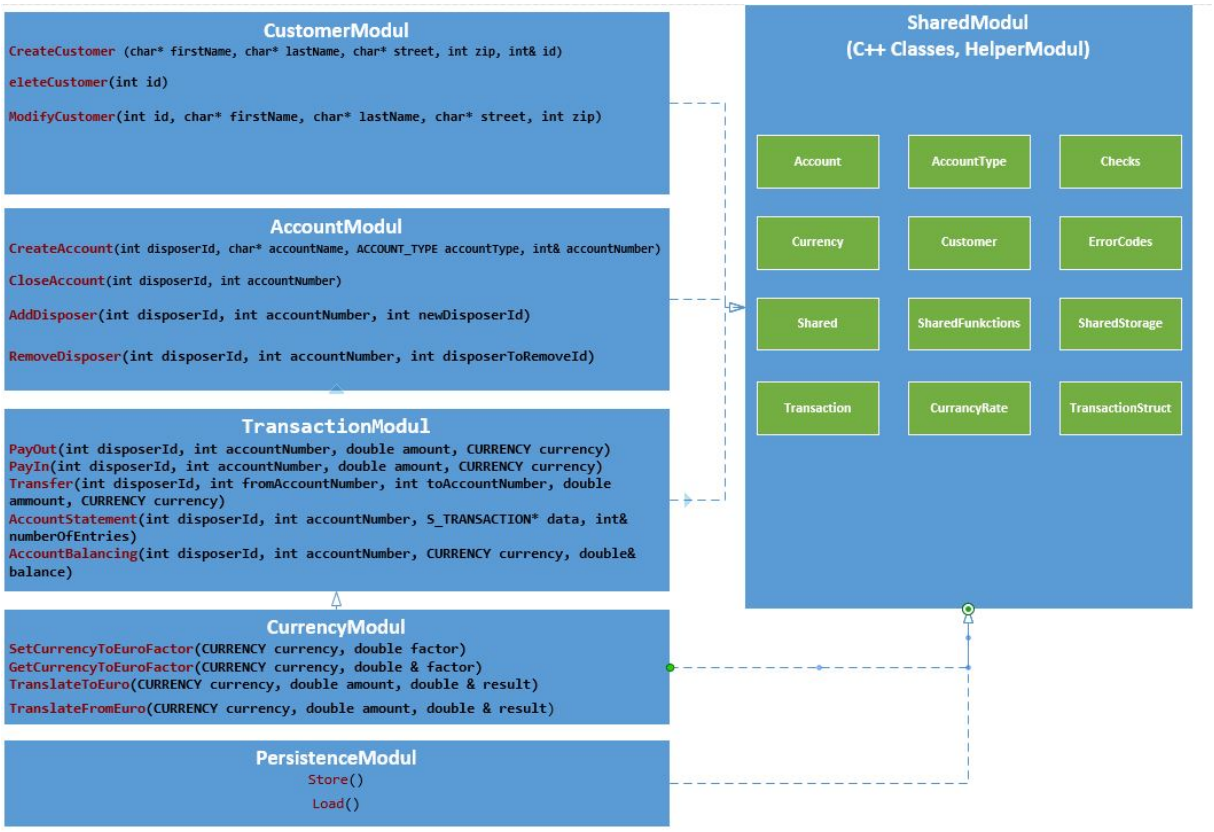

Abbildung 1: Komponentenübersicht

1.1.2 Allgemeine Hinweise

- Errocodes: Eine Liste mit Errocodes befindet sich im Kapitel 1.8
- Fehlerbehandlung (Checks) für wiederkehrende Fälle werden im Kapitel 1.9 erläutert
- Fehlerbehandlung für einzelne Funktionen werden in der Funktionsbeschreibungen erläutert

1.1.3 Funktionsbeschreibung, Komponente Kunde (CustomerModul)

Funktion:	int CreateCustomer(char* firstName, char* lastName, char* street, int zip, int& id)
Bezeichnung:	**Kunden anlegen**
Parameter:	char * firstName
Parameter:	char * lastName
Parameter:	char * street
Parameter:	int* zip
Parameter:	int & id
Rückgabewerte:	E_INVALID_PARAMETER und E_OK Siehe Kapitel 1.8 Errorcode liste)
Funktionsbeschreibung:	1) Prüfung der Übergabeparametern. Alles Strings dürfen nicht NULL sein und zip muss ein Int-Wert von 1000 bis 9999 sein. Im Fehlerfall: Rückgabewert E_INVALID_PARAMETER 2) Nach positiver Prüfung liefert die Funktion E_OK zurück und erzeugt den Kunden
Aufrufbeispiele:	CreateCustomer("Franz", "Müller", "Hintertupfing", 5020, id)

Funktion:	int DeleteCustomer(int id)
Bezeichnung:	**Kunden Löschen**
Parameter:	int id
Rückgabewerte:	E_INVALID_PARAMETER, E_CUSTOMER_NOT_FOUND und E_OK Siehe Kapitel 1.8 Errorcode liste)
Funktionsbeschreibung:	1) Prüfung der Übergabeparametern, ob id größer 0 ist. Im Fehlerfall: Rückgabewert E_INVALID_PARAMETER 2) Kunden id wird gesucht 3) Falls kein Kunde gefunden wird, oder Kunde ist Inaktiv, liefert die Funktion E_CUSTOMER_NOT_FOUND 4) Wenn id gefunden wurde, so wird der Kunde auf Inaktiv gesetzt und die Funktion liefert E_OK zurück
Aufrufbeispiele:	DeleteCustomer(id);

Funktion:	int ModifyCustomer(int id, char* firstName, char* lastName, char* street, int* zip)
Bezeichnung:	**Kunden Bearbeiten**
Parameter:	int id (Kunden- bzw. Verfüger-id)
Parameter:	char * firstName
Parameter:	char * lastName
Parameter:	char * street
Parameter:	int zip
Rückgabewerte:	E_INVALID_PARAMETER, E_CUSTOMER_NOT_FOUND und E_OK (Siehe Kapitel 1.8 Errorcode liste)
Funktionsbeschreibung:	1) Prüfung der Übergabeparametern. Hier wird nur id geprüft (muss größer oder gleich 0 sein) und im Fehlerfall wird E_INVALID_PARAMETER zurückgegeben. Restliche Werte dürfen NULL sein, so dass z.B. nur ein Parameter geändert werden kann. 2) Kunden id wird gesucht 3) Falls kein Kunde gefunden wird, liefert die Funktion E_CUSTOMER_NOT_FOUND zurück 4) Wenn alles gut geht, werden geänderte Werte übernommen und die Funktion liefert E_OK zurück
Aufrufbeispiele:	ModifyCustomer(0, "Hans", "Meier", "Daheim", &newZip);

1.1.4 Funktionsbeschreibung, Komponente Konto (AccountModul)

Funktion:	int CreateAccount(int disposerId, char* accountName, ACCOUNT_TYPE accountType, int accountNumber)
Bezeichnung:	**Konto eröffnen**
Parameter:	int disposerId
Parameter:	char* accountName
Parameter:	ACCOUNT_TYPE accountType (ein ENUM: SavingsAccount (Sparkonto) = 1; LoanAccount (Girokonto) = 2)
Parameter:	int accountNumber
Rückgabewerte:	E_INVALID_PARAMETER, E_CUSTOMER_NOT_FOUND (Siehe Kapitel 1.8 Errorcode liste)
Funktionsbeschreibung:	1) Prüfung der Übergabeparametern. Kunden- bzw. Verfügernummer muss größer oder gleich 0 und Kontoname darf nicht NULL sein. im Fehlerfall: Rückgabewert E_INVALID_PARAMETER 2) Prüfung ob die Kunden ID existiert, wenn nicht, liefert die Funktion E_CUSTOMER_NOT_FOUND 3) Wenn alles gut geht, wird ein Konto erzeugt und dem übergebenen Kunden-ID zugewiesen. Die Funktion liefert E_OK
Aufrufbeispiele:	ModifyCustomer(0,"Hans", "Meier", "Daheim", &newZip);

Funktion:	int CloseAccount(int disposerId, int accountNumber)
Bezeichnung:	**Konto schließen**
Parameter:	int disposerId
Parameter:	int accountNumber
Rückgabewerte:	E_INVALID_PARAMETER, E_REMOVE_DISPOSER_NOT_FOUND, E_ACCOUNT_NOT_FOUND, E_CUSTOMER_NOT_FOUND, E_UNAUTHORIZED, E_OK (Siehe Kapitel 1.8 Errorcode liste)
Funktionsbeschreibung:	1) Prüfung der Übergabeparametern. Kunden- bzw. Verfügernummer und Kontonummer muss größer oder gleich 0 sein. Im Fehlerfall: Rückgabewert E_INVALID_PARAMETER 2) Suche nach Konto und dazugehörige Verfüger (einen oder mehreren) 2.1) Prüfung der Übergabeparametern, im Fehlerfall: Rückgabewert E_INVALID_PARAMETER 2.2) Prüfung ob das Konto existiert oder inaktiv, wenn eins von beiden zutreffen, liefert die Funktion E_CUSTOMER_NOT_FOUND zurück 2.3) Prüfung ob Verfüger (ein oder mehrere) für das Konto autorisiert sind. Falls nicht, liefert die Funktion E_UNAUTHORIZED zurück 2.4) Falls obere Prüfungen gut gegangen sind, wird E_OK zurückgeliefert und es geht mit 3) weiter 3) Konto wird auf Inaktiv gesetzt, Zuordnung Verfüger zum Konto wird entfernt und die Funktion liefert E_OK zurück
Aufrufbeispiele:	CreateAccount(disposerId, "FirstAccount", LoanAccount, accountId); CreateAccount(0, "FirstAccount", LoanAccount, accountId);

Funktion:	AddDisposer(int disposerId, int accountNumber, int newDisposerId)
Bezeichnung:	**Verfüger zuweisen**
Parameter:	int disposerId (aktueller Kunde bzw. Verfüger)
Parameter:	int accountNumber (Kontonummer vom aktuellen Kunden)
Parameter:	int newDisposerId (neuer Kunde bzw. Verfüger)
Rückgabewerte:	E_INVALID_PARAMETER, E_NEW_DISPOSER_NOT_FOUND, E_ACCOUNT_NOT_FOUND, E_CUSTOMER_NOT_FOUND, E_UNAUTHORIZED, E_OK (Siehe Kapitel 1.8 Errorcode liste)
Funktionsbeschreibung:	1) Prüfung der Übergabeparametern. Alle drei Parameter werden müssen größer gleich 0 sein. Im Fehlerfall: Rückgabewert E_INVALID_PARAMETER 2) Suche nach Konto und dazugehörige Verfüger (einen oder mehreren) 2.1) Prüfung der Übergabeparametern, im Fehlerfall: Rückgabewert E_INVALID_PARAMETER 2.2) Prüfung ob Konto existiert oder inaktiv, wenn eins von beiden zutreffen liefert die Funktion E_CUSTOMER_NOT_FOUND 2.3) Prüfung ob Verfüger (ein oder mehrere) für das Konto autorisiert sind. Falls nicht, liefert die Funktion E_UNAUTHORIZED zurück 2.4) Falls obere Prüfungen gut gegangen sind, wird E_OK zurückgeliefert und es geht mit 3) weiter 3) Prüfung ob neue Verfüger bereits existiert, im Fehlerfall liefert die Funktion E_NEW_DISPOSER_NOT_FOUND 4) Wenn alles gut geht, wird ein neuer Verfüger zu dem übergeben Konto zugewiesen und die Funktion liefert E_OK
Aufrufbeispiele:	AddDisposer(0, 0, 33)

Funktion:	int RemoveDisposer(int disposerId, int accountNumber, int disposerToRemoveId)
Bezeichnung:	**Verfüger löschen**
Parameter:	int disposerId (Kunde bzw. Verfüger des Konotos)
Parameter:	int accountNumber (Konto von den Kunden)
Parameter:	int disposerToRemoveId (Verfüger der nicht mehr für das Konto berechtigt sein darf)
Rückgabewerte:	E_INVALID_PARAMETER, E_CANNOT_REMOVE_SELF, E_REMOVE_DISPOSER_NOT_FOUND, E_ACCOUNT_NOT_FOUND, E_CUSTOMER_NOT_FOUND, E_UNAUTHORIZED, E_OK (Siehe Kapitel 1.8 Errorcode liste)
Funktionsbeschreibung:	1) Prüfung der Übergabeparametern. Alle drei Parameter werden müssen größer gleich 0 sein. Im Fehlerfall: Rückgabewert E_INVALID_PARAMETER 2) Prüfung ob die übergeben Verfüger-ID die gleiche wie zu löschende Verfüger-ID, falls ja, liefert die Funktion Fehlercode E_CANNOT_REMOVE_SELF zurück 3) Suche nach Konto und dazugehörige Verfüger (einen oder mehreren) 3.1) Prüfung der Übergabeparametern, im Fehlerfall: Rückgabewert E_INVALID_PARAMETER 3.2) Prüfung ob Konto existiert oder inaktiv, wenn eins von beiden zutreffen liefert die Funktion E_CUSTOMER_NOT_FOUND 3.3) Prüfung ob Verfüger (ein oder mehrere) für das Konto autorisiert sind. Falls nicht, liefert die Funktion E_UNAUTHORIZED zurück 3.4) Falls obere Prüfungen gut gegangen sind, wird E_OK zurückgeliefert und es geht mit 3) weiter 4) Prüfung ob zu löschender Verfüger existiert, wenn nicht liefert die Funktion einen Fehlercode E_REMOVE_DISPOSER_NOT_FOUND zurück 5) Wenn alles gut geht, wird Zuweisung zwischen Konto und Verfüger und umgekehrt dem Verfüger das Konto entfernt. Die Funktion liefert E_OK zurück
Aufrufbeispiele:	RemoveDisposer(0, 0, 1);

1.1.5 Funktionsbeschreibung, Komponente Transaktion (TransactionModul)

Funktion:	int PayOut(int disposerId, int accountNumber, double amount, CURRENCY currency)
Bezeichnung:	**Bargeld abheben**
Parameter:	int disposerId
Parameter:	int accountNumber
Parameter:	double amount
Parameter:	CURRENCY currency (ein Enum mit allen gängigen Währungen)
Rückgabewerte:	E_INVALID_PARAMETER, E_ACCOUNT_NOT_FOUND, E_CUSTOMER_NOT_FOUND, E_UNAUTHORIZED, E_CURRENCY_FACTOR_NOT_STORED, E_INSUFFICIENT_FUNDS, E_OK (Siehe Kapitel 1.8 Errorcode liste)
Funktionsbeschreibung:	1) Prüfung der Übergabeparametern. Kunden- und Kontonummer müssen größer oder gleich 0 und Amount größer 0 sein. Im Fehlerfall: Rückgabewert E_INVALID_PARAMETER 2) Suche nach autorisierten Verfüger. E_OK falls gut gegangen, sonst: - E_INVALID_PARAMETER, falls die Daten für das Konto oder Verfüger falsch übergeben worden sind - E_ACCOUNT_NOT_FOUND, falls ungültiges Konto (NULL, oder inaktiv) - E_CUSTOMER_NOT_FOUND, falls Verfüger (Kunde) nicht gefunden (NULL, oder inaktiv) - E_UNAUTHORIZED, falls der Verfüger für das Konto nicht autorisiert ist 3) Kontostandprüfung. Rückgabe E_OK, falls gut gegangen, sonst: - E_INVALID_PARAMETER, falls die Daten für das Konto oder Verfüger falsch übergeben worden sind - Rückgabewerte aus 2), da Prüfung auf autorisierten Verfüger 4) Umrechnung der übergebenen Währung in Euro. E_OK falls gut gegangen, sonst E_CURRENCY_FACTOR_NOT_STORED, falls kein Umrechnungsfaktor für die Währung gespeichert ist

	5) Falls alles gut geht, wird die Auszahlung gespeichert und die Funktion liefert E_OK
Aufrufbeispiele:	PayOut(0, 0, 150, EUR); PayOut(0, 0, 150, USD);
Funktion:	PayIn(int disposerId, int accountNumber, double amount, CURRENCY currency)
Bezeichnung:	**Bargeld einzahlen**
Parameter:	int disposerId
Parameter:	int accountNumber
Parameter:	double amount
Parameter:	CURRENCY currency (ein Enum mit allen gängigen Währungen)
Rückgabewerte:	E_INVALID_PARAMETER, E_ACCOUNT_NOT_FOUND, E_CUSTOMER_NOT_FOUND, E_UNAUTHORIZED, E_CURRENCY_FACTOR_NOT_STORED, E_OK (Siehe Kapitel 1.8 Errorcode liste)
Funktionsbeschreibung:	1) Prüfung der Übergabeparametern. Kunden- und Kontonummer müssen größer oder gleich 0 und Amount größer 0 sein. Im Fehlerfall: Rückgabewert E_INVALID_PARAMETER 2) Suche nach autorisierten Verfüger. E_OK falls gut gegangen, sonst: - E_INVALID_PARAMETER, falls die Daten für das Konto oder Verfüger falsch übergeben worden sind - E_ACCOUNT_NOT_FOUND, falls ungültiges Konto (NULL, oder inaktiv) - E_CUSTOMER_NOT_FOUND, falls Verfüger (Kunde) nicht gefunden (NULL, oder inaktiv) - E_UNAUTHORIZED, falls der Verfüger für das Konto nicht autorisiert ist 3) Ermittlung des Faktors für die übergeben Währung. E_OK, falls gut gegangen, sonst E_CURRENCY_FACTOR_NOT_STORED, wenn der Faktor für die Währung nicht gespeichert ist 4) Falls alles gut geht, wird die Einzahlung gespeichert und die Funktion liefert E_OK zurück
Aufrufbeispiele:	PayIn(0, 0, 150, EUR); PayIn(0, 0, 150, USD);

Funktion:	int Transfer(int disposerId, int fromAccountNumber, int toAccountNumber, double amount, CURRENCY currency)
Bezeichnung:	**Überweisung**
Parameter:	int disposerId (Kunde der die Überweisung tätigt)
Parameter:	int fromAccountNumber (Kunden seine Kontonummer)
Parameter:	int toAccountNumber(Zielkonto)
Parameter:	double amount
Parameter:	CURRENCY currency (ein Enum mit allen gängigen Währungen)
Rückgabewerte:	E_INVALID_PARAMETER, E_ACCOUNT_NOT_FOUND, E_CUSTOMER_NOT_FOUND, E_UNAUTHORIZED, E_TARGET_ACCOUNT_NOT_FOUND, E_CURRENCY_FACTOR_NOT_STORED, E_INSUFFICIENT_FUNDS, E_OK (Siehe Kapitel 1.8 Errorcode liste)
Funktionsbeschreibung:	1) Prüfung der Übergabeparametern. Kunden- und seine Kontonummer müssen größer oder gleich 0 und Amount größer 0 sein. Im Fehlerfall: Rückgabewert E_INVALID_PARAMETER 2) Suche nach autorisierten Verfüger. E_OK falls gut gegangen, sonst: - E_INVALID_PARAMETER, falls die Daten für das Konto oder Verfüger falsch übergeben worden sind - E_ACCOUNT_NOT_FOUND, falls ungültiges Konto (NULL, oder inaktiv) - E_CUSTOMER_NOT_FOUND, falls Verfüger (Kunde) nicht gefunden (NULL, oder inaktiv) - E_UNAUTHORIZED, falls der Verfüger für das Konto nicht autorisiert ist 3) Prüfung des Zielkontos. E_OK falls gut gegangen, sonst E_TARGET_ACCOUNT_NOT_FOUND 4) Kontostandprüfung. Rückgabe E_OK, falls gut gegangen, sonst: - E_INVALID_PARAMETER, falls die Daten für das Konto oder Verfüger falsch übergeben worden sind - Rückgabewerte aus 2), da Prüfung auf autorisierten Verfüger 5) Umrechnung der übergebenen Währung in Euro. E_OK falls gut gegangen, sonst E_CURRENCY_FACTOR_NOT_STORED, falls kein Umrechnungsfaktor für die Währung gespeichert ist

	6) Falls das Konto von dem die Überweisung getätigt wird ein Sparkonto ist, und der Betrag kleiner als Kontostand, liefert die Funktion E_INSUFFICIENT_FUNDS zurück 7) Falls alles gut geht, wird die Überweisung abgespeichert und die Funktion liefert E_OK zurück
Funktion:	int AccountStatement(int disposerId, int accountNumber, S_TRANSACTION*** data, int& numberOfEntries)
Bezeichnung:	**Kontoauszug**
Parameter:	int disposerId
Parameter:	int accountNumber
Parameter:	S_TRANSACTION*** data (Eine Struktur mit Pointer auf Array von Transaktionen das wiederum auf die Daten zeigt)
Parameter:	int& numberOfEntries
Rückgabewerte:	E_INVALID_PARAMETER, E_ACCOUNT_NOT_FOUND, E_CUSTOMER_NOT_FOUND, E_UNAUTHORIZED, E_OK (Siehe Kapitel 1.8 Errorcode liste)
Funktionsbeschreibung:	1) Prüfung der Übergabeparametern. Kunden- und seine Kontonummer müssen größer oder gleich 0 und der Pointer nicht NULL. im Fehlerfall: Rückgabewert E_INVALID_PARAMETER 2) Suche nach autorisierten Verfüger. E_OK falls gut gegangen, sonst: - E_INVALID_PARAMETER, falls die Daten für das Konto oder Verfüger falsch übergeben worden sind - E_ACCOUNT_NOT_FOUND, falls ungültiges Konto (NULL, oder inaktiv) - E_CUSTOMER_NOT_FOUND, falls Verfüger (Kunde) nicht gefunden (NULL, oder inaktiv) - E_UNAUTHORIZED, falls der Verfüger für das Konto nicht autorisiert ist 3) Falls alles gut geht, liefert die Funktion E_OK, und ermöglicht einen Zugriff auf die S_TRANSACTION
Aufrufbeispiele:	AccountStatement(0, 0, &transactions, numOfEntries);

Funktion:	int AccountBalancing(int disposerId, int accountNumber, CURRENCY currency, double& balance)
Bezeichnung:	**Kontostand**
Parameter:	int disposerId
Parameter:	int accountNumber
Parameter:	CURRENCY currency (ein Enum mit allen gängingen Wähtungen)
Parameter:	double& balance
Rückgabewerte:	E_INVALID_PARAMETER, E_ACCOUNT_NOT_FOUND, E_CUSTOMER_NOT_FOUND, E_UNAUTHORIZED, E_CURRENCY_FACTOR_NOT_STORED, E_OK (Siehe Kapitel 1.8 Errorcode liste)
Funktionsbeschreibung:	1) Prüfung der Übergabeparametern. Kunden- und seine Kontonummer müssen größer oder gleich 0 sein. Im Fehlerfall: Rückgabewert E_INVALID_PARAMETER 2) Suche nach autorisierten Verfüger. E_OK falls gut gegangen, sonst: - E_INVALID_PARAMETER, falls die Daten für das Konto oder Verfüger falsch übergeben worden sind - E_ACCOUNT_NOT_FOUND, falls ungültiges Konto (NULL, oder inaktiv) - E_CUSTOMER_NOT_FOUND, falls Verfüger (Kunde) nicht gefunden (NULL, oder inaktiv) - E_UNAUTHORIZED, falls der Verfüger für das Konto nicht autorisiert ist 2) Kontostand wird berechnet und in die Übergebene Währung umgerechnet (weil alle Transaktionen werden in Euro gespeichert). Wenn alles gute geht, liefert die Funktion E_OK, sonst E_CURRENCY_FACTOR_NOT_STORED, falls Umrechnungsfaktor nicht gespeichert ist
Aufrufbeispiele:	AccountBalancing(0, 0, EUR, amount);

1.1.6 Funktionsbeschreibung, Komponente Währungsumrechnung (CurrencyTranslationModul)

Das Währungsmodul häng mit dem Transaktionsmodul zusammen. Sobald eine Transaktion durchgeführt wird, wird auch das Währungsmodul eingebunden.

Funktion:	int SetCurrencyToEuroFactor(CURRENCY currency, double factor)
Bezeichnung:	**Faktor für Währung zu Euro setzten**
Parameter:	CURRENCY currency (Enum)
Parameter:	double factor
Rückgabewerte:	E_INVALID_PARAMETER, E_OK (Siehe Kapitel 1.8 Errorcode liste)
Funktionsbeschreibung:	Prüfung der Übergabeparametern. Währungsfaktor darf nicht 0 sein und Währung darf nicht Eur sein. Im Fehlerfall: Rückgabewert E_INVALID_PARAMETER
Aufrufbeispiele:	SetCurrencyToEuroFactor(USD, 1);
Funktion:	GetCurrencyToEuroFactor(CURRENCY currency, double & factor)
Bezeichnung:	**Faktor Währung zu Euro ermitteln**
Parameter:	CURRENCY currency (ein Enum mit allen gängingen Wähtungen)
Parameter:	double & factor
Rückgabewerte:	E_OK, E_CURRENCY_FACTOR_NOT_STORED(Siehe Kapitel 1.8 Errorcode liste)
Funktionsbeschreibung:	falls der Faktor für die angegebene Währung nicht ermittelt werden kann, liefert die Funktion E_CURRENCY_FACTOR_NOT_STORED und sonst E_OK
Aufrufbeispiele:	GetCurrencyToEuroFactor(EUR, factor);

Funktion:	TranslateToEuro(CURRENCY currency, double amount, double & result)
Bezeichnung:	**Auf Euro umrechnen**
Parameter:	CURRENCY currency (ein Enum mit allen gängingen Wähtungen)
Parameter:	double amount
Parameter:	double & result
Rückgabewerte:	E_OK, E_CURRENCY_FACTOR_NOT_STORED (Siehe Kapitel 1.8 Errorcode liste)
Funktionsbeschreibung:	falls der Faktor für die angegebene Währung nicht ermittelt werden kann, liefert die Funktion E_CURRENCY_FACTOR_NOT_STORED und sonst wird Umgerechnet und E_OK zurügeliefert
Aufrufbeispiele:	TranslateToEuro(USD, 1, result);

Funktion:	TranslateFromEuro(CURRENCY currency, double amount, result) double
Bezeichnung:	**Von Euro aus umrechnen**
Parameter:	CURRENCY currency (ein Enum mit allen gängingen Wähtungen)
Parameter:	double amount
Parameter:	double & result
Rückgabewerte:	E_OK, E_CURRENCY_FACTOR_NOT_STORED(Siehe Kapitel 1.8 Errorcode liste)
Funktionsbeschreibung:	falls der Faktor für die angegebene Währung nicht ermittelt werden kann, liefert die Funktion E_CURRENCY_FACTOR_NOT_STORED und sonst wird Umgerechnet und E_OK zurügeliefert
Aufrufbeispiele:	TranslateFromEuro(USD, 1, result);

1.1.7 Funktionsbeschreibung, Komponente Persistenz (PersistanceModul)

Funktion:	int Store()
Bezeichnung:	**Daten speichern**
Rückgabewerte:	E_PERSISTENCE_ERROR, E_OK (Siehe Kapitel 1.8 Errorcode liste)
Funktionsbeschreibung:	Der Aufruf speichert die Daten in die shared memory (Pfad, wo sich die Komponenten befinden)

Funktion:	int Load()
Bezeichnung:	**Daten laden**
Rückgabewerte:	E_PERSISTENCE_ERROR, E_OK (Siehe Kapitel 1.8 Errorcode liste)
Funktionsbeschreibung:	der Aufruf ruft Daten, die auf shared memory zu Verfügung stehen auf und bilden eine DQL-Lite Datenbank auf

1.1.8 Errorcode Liste

pragma region OK
define E_OK 0
pragma endregion

pragma region Unexpected
define E_NOT_EXPECTED -1
pragma endregion Unexpected

pragma region Persistence
define E_PERSISTENCE_ERROR -2
pragma endregion Persistence

pragma region Authorization
define E_UNAUTHORIZED -10
pragma endregion Unauthorized

pragma region Initialize
define E_NOT_INITIALIZED -20
pragma endregion Initialize

pragma region Parameter
define E_INVALID_PARAMETER -30
pragma endregion Parameter

pragma region Customer
define E_CUSTOMER_NOT_FOUND -41
pragma endregion Customer

pragma region Account
define E_ACCOUNT_NOT_FOUND -51
define E_NEW_DISPOSER_NOT_FOUND -52
define E_REMOVE_DISPOSER_NOT_FOUND -53
define E_CANNOT_REMOVE_SELF -54
pragma endregion Account

pragma region CurrencyTranslation
define E_CURRENCY_FACTOR_NOT_STORED -60
pragma endregion CurrencyTranslation

pragma region Transaction
define E_INSUFFICIENT_FUNDS -70
define E_TARGET_ACCOUNT_NOT_FOUND -71
pragma endregion Transaction

1.1.9 Fehlerbehandlung, allgemein mögliche Übergabeparameter

Check String - ein String darf kein nullpointer sein
Check Zip - Es dürfen Werte zwischne 1000 und 9999 eingegeben werden
Check Id - die Id muss größer/gleich 0 sein
Check Currency Factor - Währungsfaktor muss größer 0 sein
Check Amount - Ein Betrag ist immer Positiv also größer 0

1.2 Funktionsweise der Implementierung von C# Assemblies (LB_02)

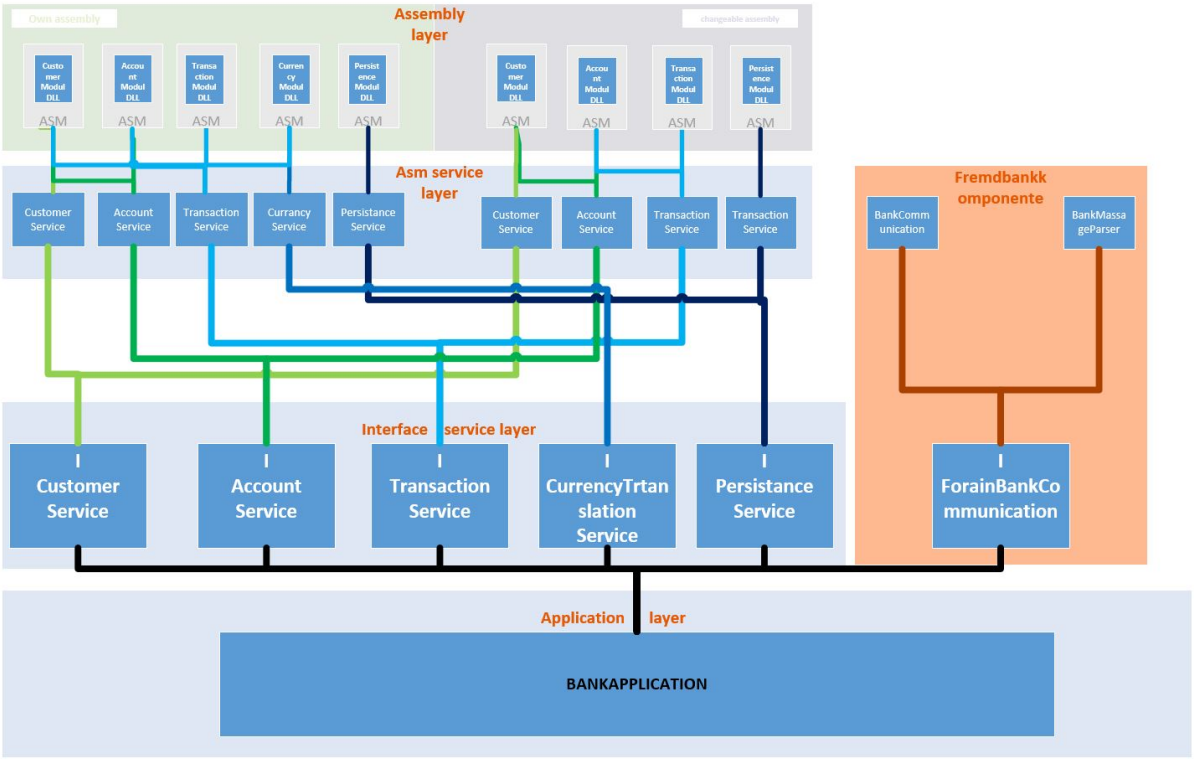

Abbildung 2: Komponentenübersicht

Die gesamte Implementierung ist auf 4 Layer aufgeteilt (Siehe Bild Komponentenübersicht).
Der **Assembley Layer** wurde in zwei Schritten erstellt:
Schritt 1) Eigene und fremde C/C++ DLL's in C# übersetzt (InternalWrapper)
Schritt 2) Die entstandene InternalWrapper in Übergeordnete Wrapper gepackt, damit diese nicht mehr über die Errorcodes, sonder Exeptions gehandelt werden.
(im Kapitel 1.3, Punkt 'Components.Common' das Exeptionhandling näher erläutert)

Im **Assembley Service Layer** ist die Anforderung von den jeweiligen Interfaces Implementiert.

Der **Interface Service Layer** stellt nun die Schnittstelle zwischen der Applikation und dem Services dar. Hier werden die geforderten Funktionalitäten nach außen abgebildet. Nach innen kennt der Interface Service Layer keine Services, sondern umgekehrt. So wurde die geforderte Kapselung umgesetzt. Zu dem definiertem Interface können beliebige Services (Komponenten) implementiert werden, so das nach außen nicht ersichtlich ist, welcher Service tatsächlich aufgerufen wird.

Im **Application layer** wurde die Applikation 'BankApplication' umgesetzt. Um Sicher zu stellen, dass alle geforderten Funktionen dem Anwender zur Verfügung stehen, gibt es in diesem Fall die Möglichkeit einen eigenen oder einen fremden Service zu laden. Der Vorteil dabei ist, dass man unabhängig von dem Fertigstellungsgrad der fremden DLL's, die volle Funktionalität an die Anwender anbieten kann!

1.2.1 Beschreibung der Assemblies und deren Schnittstellen

Grundsätzliche Erklärung zu der Beschreibung: In den Schnittstellen wird definiert, was die jeweiligen Services können müssen. Beispiel: ICustomerService hat drei Methoden (CreateCustomer, DeleteCustomer und ModifyCustomer). Alle Services (egal ob eigene oder fremde), die die Schnittstelle bedienen, müssen diese Methoden zur Verfügung stellen.
Ein Beispielaufruf wird einmalig, anhand der Komponente 'Customer Service' und der dazugehörigen Methode 'CreateCustomer' dargestellt, und kann analog bei allen anderen angewendet werden:
- Aufruf auf dem Application Layer (Der erste Aufruf gibt den zuvor gewählten Serviceprovider, fremd oder eigen zurück):

```
1 var customerService = ServiceLocator.Instance().GetService<
      ICustomerService>();
2 customerService.CreateCustomer();
```

- Aufruf auf dem Interface Service Layer:

```
1 void CreateCustomer();
```

- Aufruf auf dem Assembly Service Layer:

```
1 var id = CustomerWrapper.CreateCustomer(firstName, lastName, street, zip
      );
```

- Aufruf auf dem Assembley Layer (Wrapper):

```
1 using cw = Components.Wrapper.Own.InternalCustomerWrapper;
2 cw.CreateCustomer(firstName, lastName, street, zip, out customerId);
```

- Aufruf auf dem Assembley Layer (InternalWrapper):

```
1 internal static extern int CreateCustomer(string firstName, string
      lastName, string street, int zip, out int id);
```

- Aufruf der darunter liegenden DLL, siehe Beschreibung der Version 1.1.0

Die Beschreibung geht aus von dem 'Interface Service Layer' und von eigen (own)

Komponenten . Die Implementierung der Fremden Komponenten ist analog aufgebaut.

Die Klassen, Namespaces, Methoden und Parameter sind aus den Code-Ausschnitten ersichtlich und werden daher nicht explizit erläutert.Durch die Punkte '.....' wird signalisiert, dass es weitere Code-Teile gibt, die aber für die Beschreibung nicht relevant sind.

1.2.2 Interface Customer-Service

```
1 namespace Components.Contracts.Services
2 {
3     public interface ICustomerService
4     {
5         void CreateCustomer();
6         void DeleteCustomer();
7         void ModifyCustomer();
8     }
9 }
```

- Implementierung Service, 'Customer Service':

```
1  namespace Components.Service.Own
2  {
3      ..........
4      public class CustomerService : ICustomerService
5      {
6          public void CreateCustomer()
7          {
8              ..........
9              var id = CustomerWrapper.CreateCustomer(firstName, lastName,
                   street, zip);
10         }
11
12         public void DeleteCustomer()
13         {
14             ..........
15             CustomerWrapper.DeleteCustomer(customerId);
16         }
17
18         public void ModifyCustomer()
19         {
20             ..........
21             CustomerWrapper.ModifyCustomer(customerId, firstName,
                   lastName, street, zip);
22
```

```
23          }
24      }
25  }
```

- Wrapper 'CustomerModul':

```
1  using cw = Components.Wrapper.Own.InternalCustomerWrapper;
2  namespace Components.Wrapper.Own
3  {
4      public static class CustomerWrapper
5      {
6          public static int CreateCustomer(string firstName, string
               lastName, string street, int zip)
7          {
8          ..........
9            cw.CreateCustomer(firstName, lastName, street, zip, out
                 customerId);
10            return customerId;
11         }
12
13         public static void DeleteCustomer(int id)
14         {
15         ..........
16          cw.DeleteCustomer(id);
17         }
18
19         public static void ModifyCustomer(int id, string firstName,
               string lastName, string street, int zip)
20         {
21         ..........
22          cw.ModifyCustomer(id,firstName,lastName,street, thezip);
23         }
24     }
25 }
```

- Internal Wrapper 'CustomerModul':

```
1  namespace Components.Wrapper.Own
2  {
3      internal static class InternalCustomerWrapper
4      {
5          [DllImport(Common.DllNames.OwnCustomerModuleName,
               CallingConvention = CallingConvention.Cdecl)]
6          internal static extern int CreateCustomer(string firstName,
               string lastName, string street, int zip, out int id);
7
```

```
8           [DllImport(Common.DllNames.OwnCustomerModuleName,
                CallingConvention = CallingConvention.Cdecl)]
9           internal static extern int DeleteCustomer(int id);
10
11          [DllImport(Common.DllNames.OwnCustomerModuleName,
                CallingConvention = CallingConvention.Cdecl)]
12          internal static extern int ModifyCustomer(int id, string
                firstName, string lastName, string street, int zip);
13      }
14  }
```

1.2.3 Interface Account-Service

```
1   namespace Components.Contracts.Services
2   {
3       public interface IAccountService
4       {
5           void CreateAccount();
6           void CloseAccount();
7           void AddDisposer();
8           void RemoveDisposer();
9       }
10  }
```

- Implementierung Service, 'Account Service':

```
1   namespace Components.Service.Own
2   {
3       [Export(typeof(IAccountService))]
4       public class AccountService : IAccountService
5       {
6           public void CreateAccount()
7           {
8         ..........
9               var accountNumber = AccountWrapper.CreateAccount(disposerId,
                    accountName, accountType == AccountType.LoanAccount ?
                    Wrapper.Own.AccountType.LoanAccount : Wrapper.Own.
                    AccountType.SavingsAccount);
10          }
11
12          public void CloseAccount()
13          {
14        ..........
15              AccountWrapper.CloseAccount(disposerId, accountNumber);
16          }
17
18          public void AddDisposer()
```

```
19          {
20        ..........
21              AccountWrapper.AddDisposer(disposerId, accountNumber,
                    newDisposerId);
22          }
23
24          public void RemoveDisposer()
25          {
26        ..........
27              AccountWrapper.RemoveDisposer(disposerId, accountNumber,
                    toRemoveDisposerId);
28
29          }
30      }
31  }
```

- Wrapper 'AccountModul':

```
1   using aw = Components.Wrapper.Own.InternalAccountWrapper;
2
3   namespace Components.Wrapper.Own
4   {
5       public static class AccountWrapper
6       {
7           public static int CreateAccount(int disposerId, string
                accountName, AccountType type)
8           {
9         aw.CreateAccount(disposerId, accountName, type, out accountNumber)
              );
10              return accountNumber;
11          }
12
13          public static void CloseAccount(int disposerId, int
                accountNumber)
14          {
15            aw.CloseAccount(disposerId, accountNumber));
16          }
17
18          public static void AddDisposer(int disposerId, int accountNumber
                , int newDisposerId)
19          {
20           aw.AddDisposer(disposerId, accountNumber, newDisposerId));
21          }
22
23          public static void RemoveDisposer(int disposerId, int
                accountNumber, int disposerToRemove)
```

```
24          {
25              aw.RemoveDisposer(disposerId, accountNumber, disposerToRemove
                    ));
26          }
27      }
28  }
```

- Internal Wrapper 'AccountModul':

```
1   namespace Components.Wrapper.Own
2   {
3       internal static class InternalAccountWrapper
4       {
5           [DllImport(Common.DllNames.OwnAccountModuleName,
                CallingConvention = CallingConvention.Cdecl)]
6           internal static extern int CreateAccount(int disposerId, string
                accountName, AccountType accountType, out int accountNumber);
7
8           [DllImport(Common.DllNames.OwnAccountModuleName,
                CallingConvention = CallingConvention.Cdecl)]
9           internal static extern int CloseAccount(int disposerId, int
                accountNumber);
10
11          [DllImport(Common.DllNames.OwnAccountModuleName,
                CallingConvention = CallingConvention.Cdecl)]
12          internal static extern int AddDisposer(int disposerId, int
                accountNumber, int newDisposerId);
13
14          [DllImport(Common.DllNames.OwnAccountModuleName,
                CallingConvention = CallingConvention.Cdecl)]
15          internal static extern int RemoveDisposer(int disposerId, int
                accountNumber, int disposerToRemove);
16      }
17  }
```

1.2.4 Interface Transaction-Service

```
1  namespace Components.Contracts.Services
2  {
3      public interface IAccountService
4      {
5          void CreateAccount();
6          void CloseAccount();
7          void AddDisposer();
8          void RemoveDisposer();
9      }
10 }
```

- Implementierung Service, ' TransactionService':

```
1  namespace Components.Service.Own
2  {
3      [Export(typeof(ITransactionService))]
4      public class TransactionService : ITransactionService
5      {
6          public void PayOut()
7          {
8          ..........
9              TransactionWrapper.PayOut(disposerId, accountNumber, amount,
                   currency);
10         }
11
12         public void PayIn()
13         {
14         ..........
15             TransactionWrapper.PayIn(disposerId, accountNumber, amount,
                   currency);
16         }
17
18         public void Transfer()
19         {
20         ..........
21             TransactionWrapper.Transfer(disposerId, sourceAccountNumber,
                   targetAccountNumber, amount, currency);
22         }
23
24         public void AccountStatement()
25         {
26         ..........
27             var transactions = TransactionWrapper.AccountStatement(
                   disposerId, accountNumber);
28         }
```

```
29
30          public void AccountBalancing()
31          {
32        ..........
33              var balance = TransactionWrapper.AccountBalancing(disposerId
                    , accountNumber, currency);
34          }
35      }
36  }
```

- Wrapper 'TransactionModul':

```
1   using tw = Components.Wrapper.Own.InternalTransactionWrapper;
2
3   namespace Components.Wrapper.Own
4   {
5       public static class TransactionWrapper
6       {
7           public static void PayOut(int disposerId, int accountNumber,
                double amount, OwnCurrency currency)
8           {
9              tw.PayOut(disposerId, accountNumber, amount, currency));
10          }
11
12          public static void PayIn(int disposerId, int accountNumber,
                double amount, OwnCurrency currency)
13          {
14              tw.PayIn(disposerId, accountNumber, amount, currency));
15          }
16
17          public static void Transfer(int disposerId, int
                fromAccountNumber, int toAccountNumber, double amount,
18              OwnCurrency currency)
19          {
20              tw.Transfer(disposerId, fromAccountNumber, toAccountNumber,
                    amount,currency));
21          }
22
23          public static IEnumerable<Transaction> AccountStatement(int
                disposerId, int accountNumber)
24          {
25              ..........
26        tw.AccountStatement(disposerId, accountNumber, transactions, out
              numOfEntries));
27
28              return transactions.Select(Transaction.FromStruct).ToList();
```

```
29          }
30
31          public static double AccountBalancing(int disposerId, int
                accountNumber, OwnCurrency currency)
32          {
33          ..........
34              tw.AccountBalancing(disposerId, accountNumber, currency, out
                    balance));
35              return balance;
36          }
37      }
38  }
```

- Internal Wrapper 'TransactionModul':

```
1   namespace Components.Wrapper.Own
2   {
3       internal static class InternalTransactionWrapper
4       {
5           [DllImport(DllNames.OwnTransactionModule, CallingConvention =
                CallingConvention.Cdecl)]
6           internal static extern int PayOut(int disposerId, int
                accountNumber, double amount, OwnCurrency ownCurrency);
7
8           [DllImport(DllNames.OwnTransactionModule, CallingConvention =
                CallingConvention.Cdecl)]
9           internal static extern int PayIn(int disposerId, int
                accountNumber, double amount, OwnCurrency currency);
10
11          [DllImport(DllNames.OwnTransactionModule, CallingConvention =
                CallingConvention.Cdecl)]
12          internal static extern int Transfer(int disposerId, int
                fromAccountNumber, int toAccountNumber, double amount,
                OwnCurrency currency);
13
14          [DllImport(DllNames.OwnTransactionModule, CallingConvention =
                CallingConvention.Cdecl)]
15          internal static extern int AccountStatement(int disposerId, int
                accountNumber, [In,Out] TransactionStruct[] data, out int
                numberOfEntries);
16
17          [DllImport(DllNames.OwnTransactionModule, CallingConvention =
                CallingConvention.Cdecl)]
18          internal static extern int AccountBalancing(int disposerId, int
                accountNumber, OwnCurrency currency, out double balance);
19      }
```

```
20 }
```

1.2.5 Interface Currency-Translation-Service

```
1  namespace Components.Contracts.Services
2  {
3      public interface ICurrencyTranslationService
4      {
5          void SetCurrencyToEuroFactor();
6          void GetCurrencyToEuroFactor();
7          void TranslateToEuro();
8          void TranslateFromEuro();
9      }
10 }
```

- Implementierung Service, ' CurrancyTranslationService':

```
1  namespace Components.Service.Own
2  {
3      [Export(typeof(ICurrencyTranslationService))]
4      public class CurrencyTranslationService :
           ICurrencyTranslationService
5      {
6          public void SetCurrencyToEuroFactor()
7          {
8          ..........
9              CurrencyTranslationWrapper.SetCurrencyToEuroFactor(currency,
                   factor);
10         }
11
12         public void GetCurrencyToEuroFactor()
13         {
14         ..........
15             var factor = CurrencyTranslationWrapper.
                   GetCurrencyToEuroFactor(currency);
16         }
17
18         public void TranslateToEuro()
19         {
20         ..........
21             var result = CurrencyTranslationWrapper.TranslateToEuro(
                   currency, amount);
22         }
23
24         public void TranslateFromEuro()
25         {
26         ..........
```

```
27              var result = CurrencyTranslationWrapper.TranslateFromEuro(
                    currency, amount);
28          }
29      }
30  }
```

- Wrapper 'CurrancyTranslationWrapper':

```
1   using tw = Components.Wrapper.Own.InternalCurrencyTranslationWrapper;
2   namespace Components.Wrapper.Own
3   {
4       public static class CurrencyTranslationWrapper
5       {
6           public static void SetCurrencyToEuroFactor(OwnCurrency currency,
                double factor)
7           {
8               SaveApiCaller.ExecuteCall(() => tw.SetCurrencyToEuroFactor(
                    currency, factor));
9           }
10
11          public static double GetCurrencyToEuroFactor(OwnCurrency
                currency)
12          {
13              double factor = 0;
14              SaveApiCaller.ExecuteCall(() => tw.GetCurrencyToEuroFactor(
                    currency, out factor));
15              return factor;
16          }
17
18          public static double TranslateToEuro(OwnCurrency currency,
                double amount)
19          {
20              double result = 0;
21              SaveApiCaller.ExecuteCall(() => tw.TranslateToEuro(currency,
                     amount, out result));
22              return result;
23          }
24
25          public static double TranslateFromEuro(OwnCurrency currency,
                double amount)
26          {
27              double result = 0;
28              SaveApiCaller.ExecuteCall(() => tw.TranslateFromEuro(
                    currency, amount, out result));
29              return result;
30          }
```

```
31      }
32  }
```

- Internal Wrapper 'CurrancyTranslationWrapper':

```
1  namespace Components.Wrapper.Own
2  {
3      internal static class InternalCurrencyTranslationWrapper
4      {
5          [DllImport(DllNames.OwnCurrencyTranslationModuleName,
               CallingConvention = CallingConvention.Cdecl)]
6          internal static extern int SetCurrencyToEuroFactor(OwnCurrency
               ownCurrency, double factor);
7
8          [DllImport(DllNames.OwnCurrencyTranslationModuleName,
               CallingConvention = CallingConvention.Cdecl)]
9          internal static extern int GetCurrencyToEuroFactor(OwnCurrency
               ownCurrency, out double factor);
10
11         [DllImport(DllNames.OwnCurrencyTranslationModuleName,
               CallingConvention = CallingConvention.Cdecl)]
12         internal static extern int TranslateToEuro(OwnCurrency
               ownCurrency, double amount, out double result);
13
14         [DllImport(DllNames.OwnCurrencyTranslationModuleName,
               CallingConvention = CallingConvention.Cdecl)]
15         internal static extern int TranslateFromEuro(OwnCurrency
               ownCurrency, double amount, out double result);
16     }
17 }
```

1.2.6 Interface Persistence-Service

```
1  namespace Components.Contracts.Services
2  {
3      public interface IPersistenceService
4      {
5          void Save();
6          void Load();
7      }
8  }
```

- Implementierung Service, 'Persistance Service':

```
1  namespace Components.Service.Own
2  {
3      [Export(typeof(IPersistenceService))]
```

```
4       public class PersistenceService : IPersistenceService
5       {
6           public void Save()
7           {
8               PersistenceWrapper.Store();
9           }
10
11          public void Load()
12          {
13              PersistenceWrapper.Load();
14          }
15      }
```

- Wrapper 'PersistanceModul':

```
1   using pw = Components.Wrapper.Own.InternalPersistenceWrapper;
2
3   namespace Components.Wrapper.Own
4   {
5       public static class PersistenceWrapper
6       {
7           public static void Load()
8           {
9              pw.Load;
10          }
11
12          public static void Store()
13          {
14              pw.Stor);
15          }
16      }
17  }
```

- Internal Wrapper 'PersistanceModul':

```
1   namespace Components.Wrapper.Own
2   {
3       public static class InternalPersistenceWrapper
4       {
5           [DllImport(Common.DllNames.OwnPersistenceModule,
                CallingConvention = CallingConvention.Cdecl)]
6           public static extern int Load();
7
8           [DllImport(Common.DllNames.OwnPersistenceModule,
                CallingConvention = CallingConvention.Cdecl)]
9           public static extern int Store();
```

```
10          }
11  }
```

1.2.7 Namespace Übersicht

Components

Basisnamespace

Components.Wrapper

Alle C/C++ Dll Wrapper

Components.Wrapper.Own

Wrapper für eigene Dlls

Components.Wrapper.Foreign

Wrapper für Fremde Dlls

Components.Common

Gemeinsam genutzte Daten/Klassen/Enumerationen/Strings
Inputparser: Statische Klasse genutzt um Benutzereingaben entgegen zu nehmen, und teilweise zu validieren.
ExceptionFactory: Statische Klasse die Fehlercodes aus den eigenen C/C++ Dlls in Exceptions umwandelt
SaveApiCaller: Statische klasse, die die Aufrufe auf die Internen Wrapper in try/catch Blöcke umhüllt und aus Fehlercodes mithilfe der ExceptionFactory Exceptions generiert.

Components.Common.Exceptions

Gemeinsam genutzte Exceptions

Components.Contracts

Allgemeine Interfaces

Components.Contracts.Services

Interfaces der Services

Components.Service

Basisnamespace für die Service Implementierungen

Components.Service.Own

Implementierung der Services die die eigenen C/C++ Dlls verwenden

Components.Service.Foreign
Implementierung der Services die die fremden C/C++ Dlls verwenden

1.2.8 Zusammenfassung und Kurzbeschreibung von allen Komponenten aus dem Debug-Order

bin/Debug/AccountManagement.dll
Die C++ Dll, der anderen Gruppe, zuständig für das Account Management

bin/Debug/AccountModule.dll
Die C++ Dll, zuständig für das Account Management

bin/Debug/Components.Common.dll
dotNet Assembly wo gemeinsam genutzte Funktionen enthalten sind.

bin/Debug/Components.Contracts.dll
dotNet Assembly welche die Interfaces zu den Services (eigene und Fremde) enthält

bin/Debug/Components.Service.Foreign.AccountService.dll
Service Implementierung des fremden AccountSerivce

bin/Debug/Components.Service.Foreign.CurrencyTranslationService.dll
Service Implementierung des fremden Währungsumrechners -¿ wurde von der anderen Gruppe nicht implementiert

bin/Debug/Components.Service.Foreign.CustomerService.dll
Service Implementierung des fremden Kundenmanagements

bin/Debug/Components.Service.Foreign.PersistenceService.dll
Service Implementierung der fremden Persistence

bin/Debug/Components.Service.Foreign.TransactionService.dll
Service Implementierung des fremden Überweisungsservices

bin/Debug/Components.Service.Own.AccountService.dll
Service Implementierung des eigenen AccountSerivce

bin/Debug/Components.Service.Own.CurrencyTranslationService.dll

Service Implementierung des eigenen Währungsumrechners

bin/Debug/Components.Service.Own.CustomerService.dll
Service Implementierung des Kundenmanagements

bin/Debug/Components.Service.Own.PersistenceService.dll
Service Implementierung der Persistence

bin/Debug/Components.Service.Own.TransactionService.dll
Service Implementierung des Überweisungsservices

bin/Debug/Components.Wrapper.Foreign.AccountWrapper.dll
dotNet Wrapper der fremden C/C++ Komponente

bin/Debug/Components.Wrapper.Foreign.CustomerWrapper.dll
dotNet Wrapper der fremden C/C++ Komponente

bin/Debug/Components.Wrapper.Foreign.PersistenceWrapper.dll
dotNet Wrapper der fremden C/C++ Komponente

bin/Debug/Components.Wrapper.Foreign.TransactionWrapper.dll
dotNet Wrapper der fremden C/C++ Komponente

bin/Debug/Components.Wrapper.Own.AccountWrapper.dll
dotNet Wrapper der eigenen C/C++ Komponente. Enthält einen internen Wrapper, der nur die Aufrufe enthält und eines Wrappers der den nativen aufruf in einem try/catch block umhüllt, und die Rückgabewerte in Exceptions umwandelt.

bin/Debug/Components.Wrapper.Own.CurrencyTranslationWrapper.dll
dotNet Wrapper der eigenen C/C++ Komponente. Enthält einen internen Wrapper, der nur die Aufrufe enthält und eines Wrappers der den nativen aufruf in einem try/catch block umhüllt, und die Rückgabewerte in Exceptions umwandelt.

bin/Debug/Components.Wrapper.Own.CustomerWrapper.dll
dotNet Wrapper der eigenen C/C++ Komponente. Enthält einen internen Wrapper, der nur die Aufrufe enthält und eines Wrappers der den nativen aufruf in einem try/catch block umhüllt, und die Rückgabewerte in Exceptions umwandelt.

bin/Debug/Components.Wrapper.Own.PersistenceWrapper.dll

dotNet Wrapper der eigenen C/C++ Komponente. Enthält einen internen Wrapper, der nur die Aufrufe enthält und eines Wrappers der den nativen aufruf in einem try/catch block umhüllt, und die Rückgabewerte in Exceptions umwandelt.

bin/Debug/Components.Wrapper.Own.TransactionWrapper.dll
dotNet Wrapper der eigenen C/C++ Komponente. Enthält einen internen Wrapper, der nur die Aufrufe enthält und eines Wrappers der den nativen aufruf in einem try/catch block umhüllt, und die Rückgabewerte in Exceptions umwandelt.

bin/Debug/CurrencyTranslationModule.dll
Die C++ Dll, zuständig für die Währungsumrechnung

bin/Debug/CustomerManagement.dll
Die C++ Dll, der anderen Gruppe, zuständig für das Kundenmanagement

bin/Debug/CustomerModule.dll
Die C++ Dll, zuständig für das Kundenmanagement

bin/Debug/DataAccessLayer.dll
Die C++ Dll, der anderen Gruppe, zuständig für die Persistence

bin/Debug/PersistenceModule.dll
Die C++ Dll, zuständig für die Persistence

bin/Debug/Shared.dll
Die C++ Komponente wo gemeinsam genutzte Funktionen implementiert sind.

bin/Debug/TransactionManagement.dll
Die C++ Dll, der anderen Gruppe, zuständig für die Überweisungen

bin/Debug/TransactionModule.dll
Die C++ Dll, zuständig für die Überweisungen

1.3 Funktionsweise der Fremdbankkomponente (LB_03)

1.3.1 Einleitung

Diese Dokumentation enthält die Beschreibung der einzelnen Softwarekomponenten, die für die Kommunikation zwischen den einzelnen Bankenimplementierungen aller Laborgruppen verwendet werden kann. Umgesetzt wurden diese Komponenten als .NET Assemblies.

1.3.2 ForeignBankComponent

Die Komponente ForeignBankComponent verwendet die BankCommunikation um BankMessages zwischen der lokalen und entfernter Banken auszutauschen. Zusätzlich wird der BankMessageParser verwendet, um Nachrichten zu Serialisieren und Deserialisieren.

```
1  namespace ForeignBankComponent
2  {
3      /// <summary>
4      /// Component that encapsulates the Communication between local and
           remote bank
5      /// </summary>
6      public interface IRemoteBankComponent
7      {
8          /// <summary>
9          /// Send a Message to a remote bank.
10         /// </summary>
11         /// <param name="message">The message does specify what kind of
               message is sent.
12         /// the <see cref="Message.TransaktionsTyp"/> specifies this (
               ACK,NACK,UEBERWEISUNG,ABBUCHUNG)
13         /// </param>
14         void SendTransaction(BankMessage.Message message);
15
16         /// <summary>
17         /// The event is fired when a message is received.
18         /// There won't be another notification if a message is received
               and no one is receiving the event yet.
19         /// </summary>
20         event EventHandler<BankMessage.Message> MessageReceived;
21     }
22 }
```

Die Komponente übernimmt auch die Protokollaufgaben:
- Nachrichten die nicht erwartet werden (Hereinkommende ACK/NACK Nachrichten, wozu keine gesendete Nachricht existiert) werden verworfen.
- Nachrichten die bereits abgelaufen sind, werden automatisch mit einem NACK beantwortet.

Nachrichten werden auf der Festplatte als XML gespeichert, damit nach einem Applikationsneustart empfangene ACK Nachrichten nicht verworfen werden.

1.3.3 BankMessageParser

Die Komponente BankMessageParser stellt die Funktion der Serialisierung des Nachrichtenobjektes zur Verfügung.

```
/// <summary>
/// The serialize function. It formats the object into a single line string with a custom delimiter between with little overhead.
/// </summary>
/// <param name="message">
/// The message that gets serialized.
/// </param>
/// <returns>
/// Returns a <see cref="string"/> with the serialized text.
/// </returns>
2 Verweise | reimarklammer, vor 6 Tagen | 2 Autoren, 3 Änderungen
public static string Serialize(BankMessage message)
```


```
/// <summary>
/// The serialize function. It formats the object into a single line string with a custom delimiter between with little overhead.
/// </summary>
/// <param name="message">
/// The message that gets serialized.
/// </param>
/// <returns>
/// Returns a <see cref="string"/> with the serialized text.
/// </returns>
2 Verweise | reimarklammer, vor 6 Tagen | 2 Autoren, 3 Änderungen
public static string Serialize(BankMessage message)
```


Beschreibung der Klasse:

Die Funktion **Serialize** serialisiert den aktuellen Zustand eines BankMessage Objektes, in einem einzeiligen Text mit möglichst wenig Overhead, der über die Kommunikationsschnittstelle übertragen werden kann.

Die Funktion **Deserialize** entpackt diesen Text und gibt ein neues Objekt des Typs BackMessage zurück welches die Daten des deserializierten Textes enthält.

Verwendete Klassen:

```
1   public class Message
2   {
3       public int Version { get; set; }
4
5       public double Betrag { get; set; }
6
7       public DateTimeOffset Ablaufdatum { get; set; }
8
9       public string AbsenderBankId { get; set; }
10
11      public string EmpfaengerBankId { get; set; }
12
13      public string Waehrung { get; set; }
14
15      public string AbsenderKonto { get; set; }
16
17      public string EmpfaengerKonto { get; set; }
```

```
18
19          public TransactionType TransaktionsTyp { get; set; }
20
21          public string Verwendungszweck { get; set; }
22
23          public long MessageID { get; set; }
```

Die Klasse **BankMessage** bildet alle Informationen ab, die für den Austausch der Nachrichten zwischen den Banken benötigt werden.
Das Feld **Version** ist die Version des Protokolls.
Das Feld **Betrag** ist der Betrag der Überweisung oder der Abbuchung
Das Feld **Ablaufdatum** ist das Datum wann die Nachricht der Bank ungültig wird. Alle Nachrichten die nach diesen Datum gelesen werden, müssen von der Bank verworfen werden. Das Feld **AbsenderBankId** ist die eindeutige Identifizierung der Absenderbank
Das Feld **EmpfaengerBankId** ist die eindeutige Identifizierung der Empfaengerbank
Das Feld **Waehrung** ist die verwendete Währung.
Das Feld **AbsenderKonto** entspricht die Kontonummer des Kunden der Absenderbank.
Das Feld **EmpfaengerKonto** entspricht der Kontonummer des Kunden der EmpfängerBank.
Das Feld **TransaktionsTyp** entspricht den Typ der Nachricht. Die Typen sind wie folgt definiert:
Verwendete Klassen:

```
1  public enum TransactionType : int
2      {
3          Abbuchung = 0,
4          Ueberweisung = 1,
5          ACK = 2,
6          NACK = 3
7      }
```

Wenn es den Typ **Abbuchung** entspricht wird ein Abbuchungsauftrag von einem Kunden der Absenderbank zu einem Kunden der Empfängerbank gesendet. Diese muss dann wiederum bei Durchführung mit einer Überweisung antworten oder mit einer Nachricht mit dem Typ NACK ablehnen.

Wenn es den Typ **Überweisung** entspricht wird eine Überweisung von einem Kunden der Absenderbank zu einem Kunden der Empfängerbank gesendet. Bei Durchführung muss die Bank mit einem ACK antworten, bei Nichtdurchführung mit einem NACK.
ACK bedeutet das die Operation von der Gegenstelle genehmigt wurde.
NACK bedeutet das die Operation von der Gegenstelle abgelehnt wurde.

Das Feld **Verwendungszweck** enthält den Verwendungszweck einer Überweisung oder einer Abbuchung. Dieses optionale Feld kann auch bei einem NACK für die Fehlermeldung verwendet werden.
Das Feld **MessageID** enthält eine eindeutige Identifikationsnummer für eine Nachricht zwischen Banken. Diese Nummer wird von der Absenderbank, die einen Nachrichtenaustausch zwischen einer anderen Bank aufbaut, zufällig generiert.

1.3.4 BankCommunication

Die Komponente BankCommunication bietet die Möglichkeit zum Versenden und Empfangen von bereits serialisierten BankMessages. Diese Funktionalität wird über die Schnittstelle IBankCommunicationService veröffentlicht:

```
1  public interface IBankCommunicationService
2      {
3          /// <summary>
4          /// Event will be fired, if polling results in new messages
5          /// </summary>
6          event MessagesAvailable MessagesAvailable;
7
8          /// <summary>
9          /// Send a new message to the recipient address
10         /// </summary>
11         /// <param name="from">Sender address (e.g. an e-mail address)</
               param>
12         /// <param name="recipient">Recipient address (e.g. an e-mail
               address)</param>
13         /// <param name="message">Message payload to send</param>
14         /// <returns></returns>
15         Task Send(string from, string recipient, string message);
16
17         /// <summary>
18         /// Receive all messages that were not acknowledged yet
19         /// </summary>
20         /// <returns></returns>
21         Task<List<RawMessage>> Receive();
22
23         /// <summary>
24         /// Acknowledge message, so that it will not be in the next
               receive result.
25         /// </summary>
26         /// <param name="toAcknowledge">Raw message that should be
               acknowledged with the UID set.</param>
27         /// <returns></returns>
28         Task Acknowledge(RawMessage toAcknowledge);
```

```
29        }
30    }
```

Beschreibung des Interfaces:
Das Event **"MessagesAvailable"** wird ausgelöst, wenn nach Polling des E-Mail Eingangspostfaches neue ungelesene Mails zur Abholung bereitstehen.
Mit der asynchronen Methode **"Send"** kann eine neue Nachricht an den angegebenen Empfänger verschickt werden.
Mit der asynchronen Methode **„Receive"** können die aktuell noch nicht bestätigten Nachrichten gelesen werden.
Mit der Asynchronen Methode **„Acknowledge"** kann ein Nachrichtenempfang bestätigt werden.

Verwendete Klassen:
RawMessage:

```
1     public class RawMessage
2     {
3         /// <summary>
4         /// Unique id of the inbox for the result message
5         /// </summary>
6         public long Uid { get; set; }
7
8         /// <summary>
9         /// Payload of the raw bank Message
10        /// </summary>
11        public string Payload { get; set; }
12    }
```

UID ist die IMAP-Unique ID der Mail, wird vom IMAP Server vergeben und verwendet, um das Acknowledge auszuführen.
Payload ist der Mailbody der E-Mail-Nachricht.

Beispielverwendung und Erstellung einer IBankCommunicationService Instanz:
Die Erstellung einer IBankCommunicationService Instanz erfolgt über einen Factory.GetInstance() Aufruf. Es sind hier direct die entsprechenden SMTP und IMAP Credentials mitzugeben, damit sich die Communication Komponente mit dem richtigen Mailserver verbinden kann.
Im folgenden Codebeispiel ist die Instanzierung beispielhaft durchgeführt. Außerdem wird ein Eventhandler für das MessagesAvailable Event registriert.

```
1  namespace BankCommunicationTest
2  {
3      [TestClass]
4      public class CommunicationTests
5      {
6          [TestMethod]
7          public void Send_Success()
8          {
9              // Prepare credentials
10             var smtp = new SmtpCredentials
11             {
12                 UserName = "bank.ziegler@gmail.com",
13                 Password = "yla4DIQBKL2uZ9sFydND",
14                 Port = 587,
15                 Url = "smtp.gmail.com",
16                 UseAuthentication = true,
17                 UseSecureConnection = true
18             };
19
20             var imap = new ImapCredentials
21             {
22                 Login = "bank.ziegler@gmail.com",
23                 Password = "yla4DIQBKL2uZ9sFydND",
24                 ServerUrl = "imap.gmail.com",
25                 Port = 993,
26                 UseSsl = true,
27                 ValidateServerCertificate = true
28             };
29
30             IBankCommunicationService mailService =
                   BankCommunicationServiceFactory.
                   GetMailBasedCommunicationService(smtp, imap);
31
32             mailService.Send("bank.ziegler@gmail.com", "bank.
                   ziegler@gmail.com", "Testpayload!").Wait();
33
34             // Should be reached without exception
35             Assert.IsTrue(true);
36         }
37     }
38 }
```

1.3.5 Verwendete DLLs

Zusätzlich werden folgende Third Party Libraries verwendet:

- IMAPX: Imap Library für die Kommunikation mit den Mailkonten

2 TODO

(Alle TODO's erledigt. Topic bleibt trotzdem, falls es weitere TODO's dazukommen)

3 Offene Punkte

Die Fremde DLL's konnten nicht endgültig implementiert werden auf Grund von fehlenden Funktionalitäten aus der Laborübung 1.
Status bis zum 24.11.2016 betreffend der Implementierung der Fremd-DLL. Folgende Funktionen aus dem angebotenen Menu in der Bankapplication, beim Laden der Fremd-DLL, können nicht benutzt werden:
- RemoveDisposer()
- AddDisposer()
- SetCurrencyToEuroFactor()
- GetCurrencyToEuroFactor()
- TranslateToEuro()
- TranslateFromEuro()
- DeleteCustomer()
- ModifyCustomer()
- AccountStatement()

4 Anhang

A) Komponenten Übersicht

B) Dokumentation inkl. Komponentenübersicht

C) Die gesamte Solution

D) Video
