

DDBAC **DataDesign Banking Application Components**

Benutzerhandbuch

Übersicht

Titel	Thema	Datum
DataDesign Banking Application Components	Benutzerhandbuch	21.01.2008

Seitenumfang	Version	Status
40 Seiten	4-2-0	Freigegeben

Änderungsverzeichnis

Version	Datum	Änderungen/Kommentar
1-0-0	25.10.1999	Dokument erstellt.
2-9-0	25.09.2002	Dokument ans neue Layout angepasst. Beinhaltet ein Update zur Anpassung an Version 2.9.0 (separat beigefügte HBCI+ Dokumentation).
	07.02.2005	Customizing des Homebanking Kontakte Administrator (control panel)
3-0-7	07.02.2005	Aktualisiert zur Version 3.0.7 - DDBAC-XML Informationen hinzugefügt
4-0-0	08.08.2005	Aktualisiert zur Version 4.0.0 - DDBACBLZ Informationen hinzugefügt
	15.09.2005	Übersetzung (deutsche Version)
4-2-0	30.08.2007	Erweiterung um SEPA-Fähigkeit (siehe Abschnitt 4.9)
4-2-8	08.01.2008	Ergänzender Hinweis zum Auffinden der Beispielprogramme.

Abhängige Dokumente

Dateiname	Beschreibung
	DDBAC Referenzhandbuch
	HBCI 2.01-, HBCI 2.1-, HBCI 2.2-, FinTS 3.0-, FinTS 4.0-Spezifikationen

Inhaltsverzeichnis

1	Einleitung zur DDBAC	5
1.1	<i>Was ist DDBAC</i>	5
1.2	<i>Designkriterien</i>	5
1.3	<i>Zusammenfassung der DDBAC-Eigenschaften</i>	6
2	Entwicklung	8
2.1	<i>Runtime-Dateien</i>	8
2.2	<i>Registry-Einträge</i>	9
3	Anpassung der DDBAC (veraltet!)	9
4	Programmierung der DDBAC Applikationen	13
4.1	<i>Übersicht der DDBAC-Komponenten</i>	13
4.2	<i>DDBAC XML (FOAM)</i>	13
4.3	<i>DDBAC-Transportkomponenten</i>	13
4.3.1	<i>DDBAC-Sicherheitskomponenten</i>	13
4.3.2	<i>DDBAC CardTerminal Komponenten</i>	14
4.4	<i>DDBAC Fremdformat Komponenten</i>	14
4.4.1	<i>DDBAC Homebanking-Kontakte-Administrator</i>	14
4.5	<i>Das DDBAC Modellobjekt</i>	15
4.6	<i>Zusätzliche Objekte</i>	16
4.7	<i>Quick Start mit Visual Basic</i>	16
4.7.1	<i>Beispiel: HBCI</i>	17
4.7.2	<i>Beispiel: HBCI+</i>	18
4.7.3	<i>Zusammenfassung</i>	20
4.8	<i>Arbeit mit den DDBAC-Objekten</i>	20
4.8.1	<i>Das BACSegment-Objekt</i>	20
4.8.2	<i>Das BACCustomer-Objekt</i>	24
4.8.3	<i>Das BACDialog-Objekt</i>	25
4.8.4	<i>Behandlung von Fehlern und Ausnahmen</i>	29
4.9	<i>SEPA-Erweiterungen</i>	32
4.9.1	<i>Geschäftsvorfälle</i>	32
4.9.2	<i>SEPA-fähige Konten</i>	33
4.9.3	<i>Beispiel: Absenden einer SEPA-Nachricht</i>	34
5	Erweiterung/Ausbau der DDBAC	37
5.1	<i>Das neue HBCI-Syntaxobjektmodell</i>	37
5.2	<i>Erstellung eines neuen Security-Objektes</i>	37
6	Weitere Informationen	40
6.1	<i>Beispielsapplikationen</i>	40
6.2	<i>Segmentbeschreibungen</i>	40
6.3	<i>DDBAC Referenzhandbücher</i>	40

1 Einleitung zur DDBAC

1.1 Was ist DDBAC

Die DataDesign Banking Application Component ist eine Suite von Softwarekomponenten, die Softwarehersteller bei der Erstellung von Finanz-Applikationen (Online Electronic Banking verwendend) unterstützt.

Die DDBAC HBCI ist ein Teil der DDBAC-Architektur, das auf dem deutschen Standard „Financial Transaction Services“ (FinTS; früher: Homebanking Computer Interface: HBCI) basiert.

Bei Verwendung einer RAD-Entwicklungsumgebung, wie z.B. Visual Basic, kann eine Finanz-Anwendung innerhalb kürzester Zeit um Online-Banking erweitert werden.

Folgende Beispiele zeigen mögliche Anwendungsbereiche, in die eine Integration der DDBAC einfach möglich ist:

- Klassische Homebanking-Applikationen
- Buchhaltungssoftware (z.B. zum Lastschriftzugang oder zur automatisierten Bearbeitung der offenen-Posten-Liste, Online-Einreichung von DTA/DTAUS/DTAZV-Dateien, Accountmanagement und Account-Konsolidierung)
- POS-Software (Point-of-Sale)
- Zahlungsrouting
- Automatische Back-Office-Zahlungssysteme
- Selbstbedienungs-Bankenterminals
- Geldausgabeautomaten
- Set-Top-Box
- Brokeragesysteme
- HTML/HBCI Gateway (auf einem Web-Server laufend)

Die DDBAC realisiert die Standards HBCI 2.0, HBCI 2.0.1, HBCI 2.1, HBCI 2.2, HBCI+ (PIN/TAN erweitert), FinTS 3.0 und FinTS 4.0. Die DDBAC kann problemlos erweitert werden um bankspezifische HBCI-Segmente zu unterstützen.

1.2 Designkriterien

Die DataDesign HBCI API ist von Anfang an als leistungsfähige unabhängige Transaktionsbibliothek entwickelt worden, die in PC-Anwendungen oder Set-Top Boxen oder Organizer zum Einsatz kommen. Bei der Entwicklung der einzelnen Komponenten wurden folgende Designkriterien berücksichtigt:

Unterstützung aller möglichen Finanz-Anwendungen, die Zugang zum Online Electronic Banking benötigen. Dies wird erreicht, indem keine protokollspezifischen Ansprüche an die Applikationen gestellt und damit die Flexibilität sowie die Erweiterbarkeit maximiert werden.

Kompatibilität mit jedem HBCI/FinTS-Banksystem. Die DDBAC wird stets aktualisiert um eine größtmögliche Kompatibilität zu garantieren.

Die Erstellung von Electronic Banking Applikationen muss so einfach wie nur möglich sein. Dies wurde erreicht durch die Anwendung einer auf COM basierenden Komponentearchitektur, entworfen um in nahezu jedes RAD-Entwicklungstool zu passen.

Mit der Ausnahme vom Homebanking Kontakte Administrator (control panel applet), enthalten die DDBAC-Komponenten keine visuellen Elemente wie z.B. Eingabemasken oder Dialogboxen. Alle anderen UI-Elemente sind dem Applikationsdesigner überlassen. Das heißt, die optische Gestaltung einer Anwendung steht dem Anwendungsprogrammierer frei. Außerdem ist es so möglich die DDBAC als Basis für automatisierte Hintergrundsysteme, wie z.B. ein Abrechnungssystem das selbsttätig Banküberweisungen einreicht, zu verwenden.

Die Architektur der DDBAC basiert auf dem Microsoft Component Object Model (COM) und Win32. Als diese ist sie unmittelbar mit Windows 95, Windows 98 und Windows NT 4.0 oder späteren Versionen dieser Betriebssysteme verwendbar. Die DDBAC-Komponenten besitzen OLE Automation¹-kompatible Schnittstellen, die beinahe aus jeder Programmiersprache, einschließlich Scriptsprachen, abgerufen werden können.

1.3 Zusammenfassung der DDBAC-Eigenschaften

- Win32 COM Architektur, die virtuell alle Programmier- und Schriftsprachen unterstützt (z.B. Visual C++, Visual Basic, Visual J++, Borland Delphi, CA Visual Objects, VBScript, JScript, Perl ...).
- Unterstützt sowohl die Sicherheitsverfahren RDH und DDV, spezifiziert durch HBCI oder FinTS, als auch PIN/TAN, spezifiziert durch HBCI+ (HBCI PIN/TAN Erweitert) oder FinTS. RDH wird unterstützt durch eine reine, auf Software (RDH-Diskette) basierende Lösung und durch RSA SmartCards. Der SmartCard-Zugriff wird unterstützt durch alle CT-API- oder PC/SC-konforme SmartCard-Leser.
- Unterstützt alle Transaktionstypen definiert in HBCI 2.0, HBCI 2.0.1, HBCI 2.1, HBCI 2.2, FinTS 3.0 und FinTS 4.0. Die DDBAC kann leicht um zusätzliche Transaktionstypen erweitert werden, ohne den existierenden Code neu übersetzt werden muss.
- Die DDBAC enthält Komponenten um fremde in HBCI oder FinTS benutzte Datenformate zu verarbeiten: S.W.I.F.T. MT-940 (Kontoauszug), MT-942 (vorgemerkte Umsätze), MT-571 (Depotaufstellung) sowie Entwicklung des DTAUS (Sammelüberweisung und Sammellastschrift) und DTAZV (Auslandsüberweisung).
- Unterstützt transparente Dateiübertragung.

¹ Früher wurden sie von der Microsoft Marketingabteilung als ActiveX Components bezeichnet. Später wurde diese Technologie in Automation umbenannt. In diesem Dokument wird der Originalname OLE Automation verwendet.

- Unterstützt TCP/IP- und HTTP/HTTPS-Kommunikation. Andere Kommunikationstypen (BACTransport) können leicht hinzugefügt werden.
- Enthält einen "Homebanking Kontakte Administrator" als Control Panel Applet. Dieses Applet unterstützt die Assistentenbasierte Einrichtung und Verwaltung von HBCI/FinTS-Kontakten. Damit bleibt es dem Anwendungshersteller erspart, diese Funktionalität in die Anwendungssoftware zu integrieren.
- Unterstützt HBCI+ (HBCI PIN/TAN Erweitert).
- Unterstützt das zwei-Schritt-TAN-Verfahren für PIN/TAN, spezifiziert in FinTS 3.0 (für beide Prozessvarianten).
- Unterstützt im vollen Umfang FinTS 3.0 (PIN/TAN) und FinTS 4.0 (PIN/TAN) sowie FinTS 3.0 (RDH-1) und FinTS 4.0 (RDH-1).

2 Entwicklung

2.1 Runtime-Dateien

Die DDBAC besteht aus folgenden Runtime-Dateien. Für eine lauffähige Version der DDBAC auf dem Zielsystem müssen folgende Dateien installiert werden.

Dateiname	Datei-Version	Beschreibung
DDBAC.DLL	4,0,0,0	DataDesign DDBAC Komponentenrahmen
DDBACASP.CPL	4,0,0,0	DataDesign DDBAC WinHTTP Transportkomponente ²
DDBACBLZ.DLL	4,0,0,0	DataDesign DDBAC Bankleitzahlen (BLZ)
DDBACCPL.CPL	4,0,0,0	DataDesign DDBAC Homebanking-Kontakte-Administrator
DDBACCTM.CPL	4,0,0,0	DataDesign DDBAC CardTerminal Systemsteuerung
DDBACCT.DLL	4,0,0,0	DataDesign DDBAC CT-API CardTerminal Komponente
DDBACDDV.DLL	4,0,0,0	DataDesign DDBAC DDV ChipCard-Sicherheitskomponente
DDBACDT2.DLL	4,0,0,0	DataDesign DDBAC DTAUS und DTAZV Formatkomponente
DDBACHTP.DLL	4,0,0,0	DataDesign DDBAC HTTP Transportkomponente
DDBACICC.DLL	4,0,0,0	DataDesign DDBAC RSA Chipcard-Sicherheitskomponente
DDBACMT9.DLL	4,0,0,0	DataDesign DDBAC S.W.I.F.T. Formatkomponente
DDBACRDH.DLL	4,0,0,0	DataDesign DDBAC RDH Sicherheitskomponente
DDBACSC.DLL	4,0,0,0	DataDesign DDBAC PC/SC SmartCard API Komponente
DDBACTAN.DLL	4,0,0,0	DataDesign DDBAC HBCI+ Komponente
DDBACTCP.DLL	4,0,0,0	DataDesign DDBAC TCP/IP Transportkomponente
DDBACXML.DLL	4,0,0,0	DataDesign DDBAC FOAM Dienstleister fürs HBCI
DDGK.DLL	4,0,0,0	DataDesign DDBAC Geldkartekomponente

Für weitere Informationen siehe [ReadMe.html](#).

² Nicht in übliche Setups eingefügt. Diese Datei ist nur für Server-Setups verwendet. Bei weiteren Fragen wenden Sie sich bitte an Ihren Ansprechpartner der DataDesign AG.

2.2 Registry-Einträge

Für eine typische Installation der DDBAC-Dateien sind keine Registry-Einträge erforderlich. Es gibt dennoch mehrere optionale Registry-Werte, die bei OEM-Installationen oder durch Systemadministratoren verwendet werden können, um die DDBAC zu konfigurieren.

Alle OEM Registry-Einträge sind String-Werte innerhalb des Keys [HKEY_CURRENT_USER\SOFTWARE\DataDesign\DDBAC] und [HKEY_LOCAL_MACHINE\SOFTWARE\DataDesign\DDBAC] (sofern für den User dafür Schreibrechte bestehen). Die nachfolgende Tabelle zeigt die unterstützten Registry-Werte, deren Fehlwert und Beschreibung.

Bis Version 3.0.6.6 wurden Registry-Einträge ausschließlich unter [HKEY_LOCAL_MACHINE\SOFTWARE\DataDesign\DDBAC] abgespeichert.

Name des Wertes	Beschreibung
InstallDir	Falls nicht vorhanden, wird das Standardverzeichnis verwendet (%CommonProgramFiles%\DataDesign\DDBAC). InstallDir gibt das Verzeichnis an, in das die Runtime-Dateien installiert werden.
DataDir	Falls nicht vorhanden, wird unter %AppData% (d.h. SHGetSpecialFolderPath mit CSIDL_APPDATA) das Unterverzeichnis "DataDesign\DDBAC" erstellt und verwendet. DataDir wird zum Abspeichern der Datei DDUSERS.DAT sowie aller UPD- und BPD-Dateien verwendet. Man kann das Verzeichnis auf ein gemeinsames Netzwerk umleiten, um die Dateien gemeinsam mit mehreren Arbeitsplätzen zu nutzen. Die gemeinsame Nutzung kann jedoch zu einem Sicherheitsrisiko führen, da mehrere Nutzer auf die Kontoinformationen zugreifen könnten. Aber Version 4.0 besteht die Möglichkeit die Daten der DDUSERS.DAT sowie der UPD's und BPD's in einem eigenen Datencontainer abzuspeichern (siehe BACStorage).
PersistBPD	Der Standardwert "1" bewirkt, dass alle neu eingehenden BPD-Dateien in DataDir abgespeichert werden. Wenn dieser Wert auf "0" gesetzt wird, werden keine BPD-Dateien abgespeichert (z. B. in gemeinsam verwendeten Serverumgebungen).
PersistUPD	Der Standardwert "1" bewirkt, dass alle neu eingehenden UPD-Dateien in DataDir abgespeichert werden. Wenn dieser Wert auf "0" gesetzt wird, werden keine UPD-Dateien abgespeichert (z. B. in gemeinsam verwendeten Serverumgebungen).
ResponseTimeout	Standardwert: "120000". Gibt einen response timeout von zwei Minuten(120000 ms) an. Bei Bedarf kann es auf einen anderen Wert umgestellt werden.
IsTraceOn	Standardwert "0" bewirkt, dass kein Logging stattfindet. Der Wert kann im Tab "Herstellerhinweis" im Homebanking-Kontakte-Administrator geändert werden. Wenn das Logging eingeschaltet wird, beträgt der Wert „1“ und alle Nachrichten werden auf der Datei "%USERPROFILE%\HBCILog.txt" gespeichert.
CustomizeUI (als Key)	Veraltet: Beinhaltet den Namen eines Registry-Keys, der für das angepasste Layout (Customizing) des Homebanking-Kontakte-Administrator verwendet wird (siehe auch 3 Anpassung der DDBAC). Falls nicht vorhanden, wird das Standard-Layout des Homebanking-Kontakte-Administrator benutzt.

3 Anpassung der DDBAC (veraltet!)

Die folgende Information über die Anpassung der DDBAC ist veraltet.

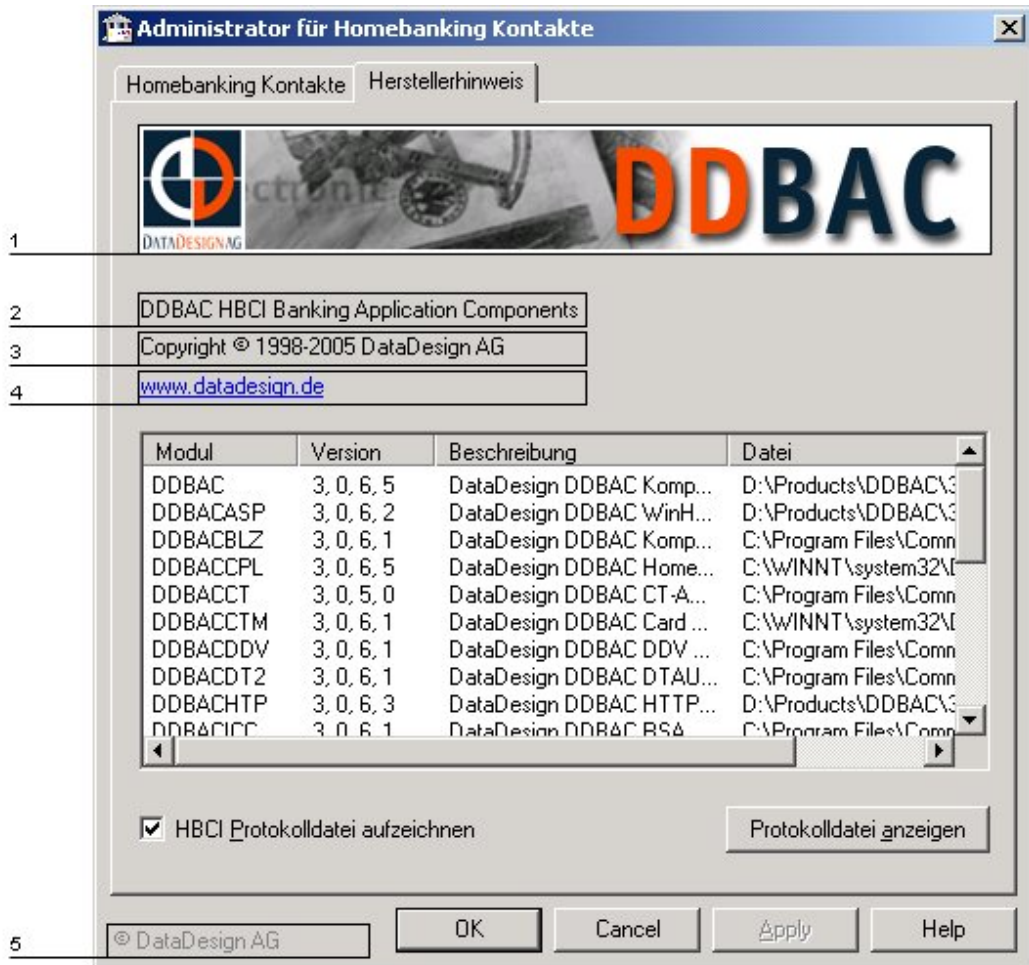
In der endgültigen Version der DDBAC 4.0 wird (zusätzlich) ein völlig neuer "Homebanking-Kontakte-Administrator" sowie neue Funktionalitäten zu dessen Anpassung verfügbar sein.

Wie bereits erwähnt, sind die einzigen von der DDBAC unterstützten UIs (User-Interfaces) zwei Control Panel Applets ("Homebanking Kontakte" und "Chipkartenleser"). Mit eigenen Bitmaps kann man das Layout des "Homebanking-Kontakte-Administrator" beeinflussen. Dies ist möglich seit der DDBAC-Version 3,0,6,x. Bei früheren Versionen haben diese Erweiterungen keinen Effekt.

Wird der "Homebanking-Kontakte-Administrator" aus der Systemsteuerung heraus aufgerufen, hat das Customizing ebenfalls keine Auswirkungen. In diesem Fall wird das Standard-Layout verwendet.

Zuerst muss der Registry-Key "CustomizeUI" (siehe auch 2.2 Registry-Einträge) unter [HKEY_CURRENT_USER\SOFTWARE\DataDesign\DDBAC] erstellt werden. In diesen Key müssen die unten beschriebenen Registry-Werte eingetragen werden werden.

Die nächsten beiden Screenshots zeigen die entsprechenden Möglichkeiten auf:



Name des Wertes	Beschreibung
ImgAbout	(1) Verweist auf eine .bmp-Datei (mit vollständiger Pfadangabe). Das Bitmap muss eine Größe von 435 x 65 Pixel haben. Es gibt keine Grenzen für die Anzahl der Farben. Es muss jedoch beachtet werden, dass auf manchen Systemen nicht alle Farben gezeigt werden können.
Copyright1	(2) Beinhaltet den in der ersten Zeile angezeigten Text. Die Änderung der Copyright-Information durch diesen Wert ist nur dann erlaubt, wenn Ihre Vertragsvereinbarungen ein entsprechendes Entfernen der DataDesign Copyright-Information zulassen!
Copyright2	(2) Beinhaltet den in der zweiten Zeile angezeigten Text. Die Änderung der Copyright-Information durch diesen Wert ist nur dann erlaubt, wenn Ihre Vertragsvereinbarungen ein entsprechendes Entfernen der DataDesign Copyright-Information zulassen!
CopyrightLink	(4) Beinhaltet einen http://-Link (als Text), der in der dritten Zeile angezeigt wird und auf die entsprechenden Seiten verlinkt.
ShowCopyright	(5) Falls Sie die DataDesign Copyright-Information nicht zeigen wollen, stellen Sie den Wert auf "no" (als Text). Falls dieser Wert auf "no" gesetzt ist, werden die DataDesign Copyright-Information weder in den Online-Dialogfenstern (während Senden/Empfangen einer Information an/vom Bankserver) noch allen anderen Dialogfenstern angezeigt. Das Entfernen der Copyright-Information hierdurch ist nur dann erlaubt, wenn Ihre Vertragsvereinbarungen dies zulassen!
ImgNewContact	(6) Verweist auf eine .bmp-Datei (mit vollständiger Pfadangabe). Das Bitmap muss eine Größe von 120 x 226 Pixel haben. Es gibt keine Grenzen für die Anzahl der Farben. Es muss jedoch beachtet werden, dass auf manchen Systemen nicht alle Farben gezeigt werden können.

Ein zusätzliches Beispiel mit allen oben angeführten Registry-Keys im .reg-Datei-Format (für DDAG als Beispiel):

```
[HKEY_LOCAL_MACHINE\SOFTWARE\DataDesign\DDBAC\DDAG]

"ShowCopyright"="NO"
"Copyright1"="Zeile1"
"Copyright2"="Zeile2"
"CopyrightLink"="http://www.datadesign.de"
"CopyrightLinkText"="DataDesign AG"
"ImgNewContact"="C:\\Wizard.bmp"
"ImgAbout"="C:\\Banner.bmp"
```

Um den "Homebanking-Kontakte-Administrator" im eigenen Layout zu starten, muss die Option "CustomizeUI" eingestellt werden. Diese kann erfolgen durch `BACBanking.Options["CustomizeUI"]="DDAG"` (für DDAG als Beispiel; bitte verwenden Sie anstelle von DDAG Ihren eigenen Key-Namen).

Nach dem Einstellen der CustomizeUI-Option kann der "Homebanking-Kontakte-Administrator" mit `BACBanking.RunContactsCPL` oder `BACBanking.RunNewContactWizard` gestartet werden.

4 Programmierung der DDBAC Applikationen

4.1 Übersicht der DDBAC-Komponenten

Die DDBAC Komponenten bestehen aus einem eng gekoppelten Kernsystem, dem DDBAC Framework, sowie verschiedene lose angekoppelte Dienstkomponenten. Das komplette Kernsystem ist physikalisch in der DDBAC.DLL untergebracht. Die Dienstkomponenten sind normalerweise in eigenen DLL Dateien untergebracht und somit leicht austauschbar.

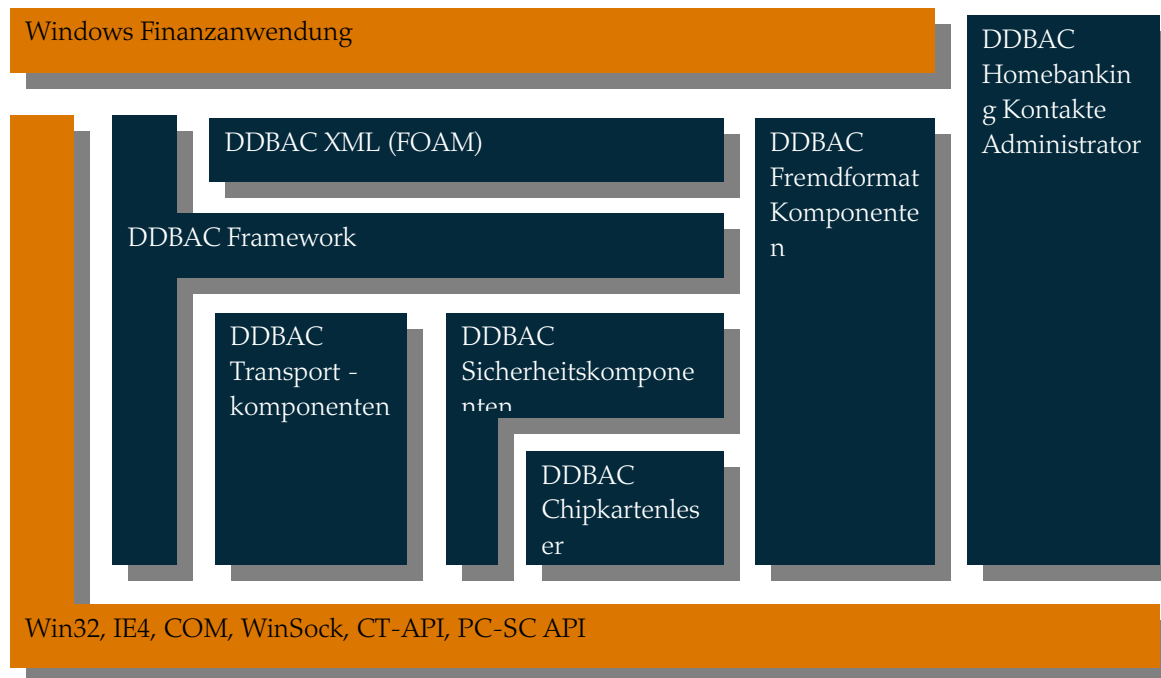


Abb. 1 –DDBAC Architektur

4.2 DDBAC XML (FOAM)

Es gibt zwei Möglichkeiten auf die DDBAC zuzugreifen. Entweder, man benutzt direkte API calls oder XML (FOAM-XML) mit nur drei unterschiedlichen Schnittstellen. Während man XML benutzt, können die Auftragsdaten als well-formed XML-Daten (FOAM-XML) gesendet werden. Die FOAM-Spezifikation ist frei verfügbar, ein Applikationsbeispiel (wie HBCIPAD) wird ebenfalls in Kürze verfügbar sein. Bei weiteren Fragen wenden Sie sich bitte an Ihren Ansprechpartner der DataDesign AG.

4.3 DDBAC-Transportkomponenten

Als Standardkommunikation wird TCP/IP über Internet gemäß HBCI Standard unterstützt. Neue Kommunikationsdienste (z.B. zusätzlich BTXFiF) können vom Anwendungsprogrammierer als Komponenten in die DDBAC integriert werden. Die Verarbeitung der HBCI Nachrichten in den DDBAC ist vollständig unabhängig vom verwendeten Transportmedium.

4.3.1 DDBAC-Sicherheitskomponenten

Es werden alle gängigen Sicherheitsmedien im symmetrischen und asymmetrischen Verfahren unterstützt:

- Disketten basierendes RDH Verfahren (Software Kypto)

- Chipkarten basierendes Verfahren mit DDV-Karte (Sparkassen-version)
- Chipkarten basierendes Verfahren mit RSA-Karte (Private Banken – BdB)

Alle implementierten Kryptoverfahren sind frei exportierbar und unterliegen keinerlei Exportbeschränkungen. (768 BIT RSA mit eigenen europäischen Bibliotheken).

Die in den DDBAC verwendeten Sicherheitsverfahren werden durch externe HBCI Security Provider Komponenten implementiert. Die Schnittstelle zu diesen Komponenten ist offen gelegt. Das ermöglicht dem Anwendungsprogrammierer zusätzlich zu den standardmäßig unterstützten Sicherheitsverfahren, eigene Sicherheitsverfahren zu entwickeln und in die DDBAC zu integrieren.

Mit DDBAC 4.0 wird auch die ZKA-Chipkarte (Signaturkarte) unterstützt.

4.3.2 DDBAC CardTerminal Komponenten

Auf einer ChipCard basierende Sicherheitsverfahren greifen auf den Chipkartenleser durch die DDBAC CardTerminal Komponenten zu. Es sind zwei Standard CardTerminal Komponenten enthalten. Die erste unterstützt alle CT-API-konforme SmartCard-Leser (DDBACCT.DLL) und die zweite alle PC/SC SmartCard API-konforme Leser (DDBACSC.DLL). Bei Bedarf kann die DDBAC so durch den Anwendungsprogrammierer um zusätzliche CardTerminal Komponenten, z.B. um eine proprietäre SmartCard-Leser API zu unterstützen, erweitert werden.

4.4 DDBAC Fremdformat Komponenten

In HBCI/FinTS transparent eingestellte Fremdformate, wie z.B. S.W.I.F.T. und DTAUS werden nahtlos durch separat implementierte Komponenten integriert. Standardmäßig sind Komponenten für die folgenden Formate vorhanden:

- S.W.I.F.T. MT-940 ("Kontoauszug")
- S.W.I.F.T. MT-942 ("Nicht gebuchte Transaktionen")
- S.W.I.F.T. MT-571 (Depotübersicht)
- S.W.I.F.T. MT-535 ("Depotauszug, neu")
- DTAUS (Sammelüberweisung und Sammellastschrift)
- DTAZV (Auslandsüberweisungen)
- BLOB (Transparenter Datenblock, nutzbar für Filetransfers)

Die Unterstützung für weitere Formate kann vom Anwendungsprogrammierer durch eigene Komponentenerweiterungen realisiert werden.

4.4.1 DDBAC Homebanking-Kontakte-Administrator

Die aufwendigen und hochgradig institutsspezifischen Funktionen zur Einrichtung und Verwaltung von HBCI Zugängen wird vom DDBAC Homebanking-Kontakte-Administrator zentral in der Windows Systemsteuerung übernommen. Eine Finanzanwendung kann sich somit auf seine eigentliche Funktionalität beschränken.

Mit Hilfe der DDBAC HBCI Zugangsverwaltung kann der Endkunde seine HBCI Zugänge und Sicherheitsmedien wie z.B. Chipkarte und Chipkartenlesegerät einrichten und Verwaltungsaufgaben wie z.B. das Sperren des Zuganges wahrnehmen.

4.5 Das DDBAC Modellobjekt

Üblicherweise verschafft sich der Anwendungsprogrammierer einen Überblick über eine Komponentenbibliothek mit Hilfe des zugrunde liegenden Objektmodells. Das Objektmodell stellt die Komponenten und Ihre Abhängigkeiten in einer stark vereinfachten Form dar. Das Objektmodell gibt keine Auskunft über die physikalische Struktur, d.h. darüber wie die einzelnen Objekte in Moduldateien gruppiert sind. Für den Anwendungsprogrammierer ist dies auch nicht relevant.

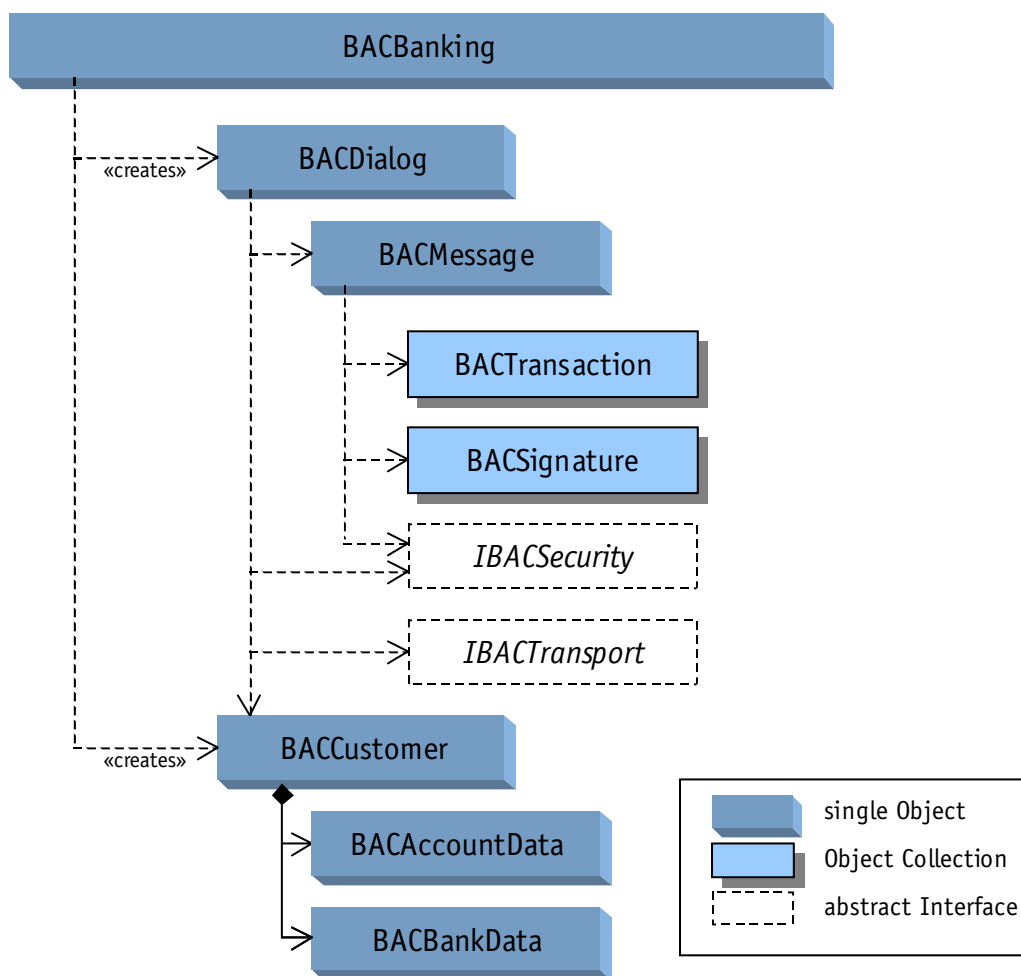


Abb. 2 – Modellobjekt

Das in der Abbildung gezeigte Objektmodell gibt nur die Objekte wieder, die für die Gesamtstruktur der DDBAC von Bedeutung sind. So wurden das BACSegment Objekt sowie die Objekte zur Verarbeitung von Fremdformaten weggelassen.

Die einzelnen Objekte in diesem Objektmodell haben folgende Aufgaben:

Objekt	Zweck
BACBanking	Dies ist das primäre Objekt der DDBAC. Eine Anwendung wird immer mindestens eine Instanz dieses Objektes erzeugen um die DDBAC zu verwenden. Das BACBanking Objekt stellt allgemeine Verwaltungs- und Hilfsfunktionen zur Verfügung. Die BACDialog und BACCustomer Objekte können nur über das BACBanking Objekt und nicht direkt instantiiert werden.
BACCustomer	Ist verantwortlich für die Verwaltung und Speicherung aller Informationen zu den einzelnen Kunde-Bank Beziehungen. Eine BACCustomer Instanz repräsentiert genau eine Kunde-Bank Beziehung mit all ihren Attributen. Eine BACCustomer Instanz wird eindeutig identifiziert durch die Kombination aus Länderkennzeichen (CountryCode), Bankleitzahl (BankID) und Benutzerkennung (UserID).
BACDialog	Das BACDialog Objekt verwaltet die Online-Sitzung. Das heißt es ist für das korrekte Senden und Empfangen von HBCI Nachrichten (BACMessage) im HBCI Dialog Kontext verantwortlich.
IBACTransport	Dies ist ein Platzhalter für eine beliebige Implementierung einer Kommunikationskomponente. Eine bestimmte Implementierung dieser Objektkategorie wird normalerweise als eigenständiges Modul (eigene DLL) realisiert.
IBACSecurity	Dies ist ein Platzhalter für eine beliebige Implementierung einer DDBAC Security Komponente. Diese Komponente stellt den Sicherheitskontext für den Dialog zur Verfügung.
BACAccountData	Für jeden Benutzer existiert ein BACAccountData Objekt das die Informationen zu allen von ihm verwalteten Konten vorhält. Dieses Objekt entspricht den HBCI User Parameter Daten (UPD). Haben mehrere Benutzer eines Systems Zugriff auf dasselbe Konto, werden die zugehörigen Informationen getrennt verwaltet. Es ist die Aufgabe der Anwendung diese getrennten Informationen zu verknüpfen.
BACBankData	Für jede Bank existiert ein BACBankData Objekt das die allgemeinen Informationen über diese Bank vorhält. Dieses Objekt entspricht den HBCI Bank Parameter Daten (BPD).
BACMessage	Repräsentiert eine Homebanking-Nachricht und die zugehörige Bankantwort. Eine Nachricht ist die Einheit einer Online Abwicklung. Jede Nachricht kann eine oder mehrere Transaktionen (BACTransaction) enthalten.
BACTransaction	Ein BACTransaction Objekt speichert und verwaltet die gesamten Information zu einem Bankauftrag. Die Anwendung legt für jeden Auftrag ein neues BACTransaction Objekt an und initialisiert es mit allen für diesen Auftrag notwendigen Parametern. Das Objekt wird dann zur Bearbeitung im Rahmen eines Online Dialoges in eine BACMessage eingestellt. Alle Statusinformationen, Fehlermeldungen, sowie Ergebnisse die sich auf diesen Auftrag beziehen werden ebenfalls in diesem BACTransaction Objekt abgelegt.
BACSignature	Ein BACSignature-Objekt enthält eine einzelne Benutzersignatur.
BACSyntax	Das BACSyntax-Objekt enthält Informationen über die Syntax aller unterstützten Segmente. Benutzen Sie dieses Objekt um neue Segmente hinzuzufügen oder um nach der Syntax eines bereits bestehenden abzufragen.
BACSegment	Das Arbeitspferd der DDBAC. Hilfsobjekt das einen einzelnen HBCI Datensatz darstellt. Es ist dieses Objekt das für das Generieren und Parsen von HBCI Segmenten verantwortlich ist.

4.6 Zusätzliche Objekte

Objekt	Zweck
BACBLZInfo	Die DDBAC enthält alle detaillierten Informationen über die BankServer (die Zugangs-Datenbank), die benutzten Sicherheitsmethoden und der HBCI-/FinTS-Version(en), etc. Das Objekt BACBLZInfo erlaubt eine Suche innerhalb dieser Datenbank.

4.7 Quick Start mit Visual Basic

Dieser Abschnitt enthält einige Code-Beispiele, die eine Verwendung der DDBAC demonstrieren sollen. Es wird auf beide Sicherheitsmedien (HBCI und HBCI+) eingegangen.

Weitere Hinweise zu Beispielen finden Sie im Abschnitt 6.1 (Beispielsapplikationen) in diesem Dokument.

4.7.1 Beispiel: HBCI

Das folgende Beispiel zeigt die komplette Abwicklung eines HBCI-Dialogs mittels eines übersichtlichen VBScript-Programms auf. Auch wenn Sie kein VBScript einsetzen, sollte es Ihnen keine Probleme bereiten, dem Beispiel zu folgen. Das Beispiel holt den Saldo des ersten Kontos des ersten eingerichteten Kontakts vom Server ab und zeigt diesen an.

```

Dim objBanking ' As BACBanking
Dim objCustomer ' As BACCustomer
Dim objDialog ' As BACDialog
Dim sPIN ' As String
Dim objHKSAL ' As BACSegment
Dim objHIUPD ' As BACSegment
Dim objHISAL ' As BACSegment
Dim objMessage ' As BACMessage
Dim sSaldo ' As String

' Zuerst wird das Basis-Objekt der DDBAC, das BACBanking-Objekt, erstellt.
' Das BACBanking-Object ist ein "monostate"-Object, das heißt, dass es
' keinen local state besitzt. Alle Instanzen dieses Objektes sind identisch.
' Von dieser Basis kann auf die Collection BACContact zugegriffen werden. In diesem
' Beispiel greifen wir auf den allerersten Eintrag aus dieser Kollektion.

Set objBanking = CreateObject("DataDesign.BACBanking")
Set objCustomer = objBanking.Customers(0)

' Um einen authentifizierten Dialog zu beginnen, muss der Kunde sein
' Sicherheitsmedium und seine PIN eingeben. Anhand dieser Information
' kann der Dialog starten.

sPIN = InputBox( _
    "Bitte geben Sie Ihre PIN bzw. Ihre Passphrase für den HBCI Kontakt " & _
    objCustomer.Fields("Contact") & " ein")
Set objDialog = objCustomer.NewDialog(sPIN)
objDialog.BeginDialog 1, "", Nothing

' Erstellt ein HKSAL (Saldo) Auftragssegment. Für dieses Beispiel greifen
' wir auf das allererste Konto aus den für den Kunden verfügbaren Konten.
' Für jedes Konto ist ein HIUPD-Segment in der Segmentkollektion des
' AccountData-Objektes gespeichert.

Set objHIUPD = objCustomer.AccountData.Segments(0)
Set objHKSAL = objBanking.NewSegment("HKSAL", 3)
objHKSAL("AuftraggeberKontoverbindung1", "Kontonummer1") = _
    objHIUPD("Kontoverbindung1", "Kontonummer1")
objHKSAL("AuftraggeberKontoverbindung1", "Laenderkennzeichen1") = _
    objHIUPD("Kontoverbindung1", "Laenderkennzeichen1")
objHKSAL("AuftraggeberKontoverbindung1", "Kreditinstitutcode1") = _
    objHIUPD("Kontoverbindung1", "Kreditinstitutcode1")
objHKSAL("AlleKonten1") = False

```

```
objHKSAL("Kontowaehrung1") = objHIUPD("Kontowaehrung1")

' Ausführung der Transaktion und Beendung des Dialogs. Die Antwort wird
' durch das BACMessage-Objekt aufgefangen.
Set objMessage = objDialog.ExecuteSegment(objHKSAL)
objDialog.EndDialog Nothing

' Auswertung der Antwort. Wir überprüfen hier keine Fehler, sondern
' greifen auf das allererste Segment aus der ResponseSegments-Kollektion.
Set objHISAL = objMessage.Transactions(0).ResponseSegments(0)

sSaldo = objHISAL("SaldoGebucht1", "Waehrung1") & " "
If objHISAL("SaldoGebucht1", "SollHabenKennzeichen1") = "D" Then
    sSaldo = sSaldo & "-"
End If
sSaldo = sSaldo & objHISAL("SaldoGebucht1", "Wert1")

MsgBox "Aktueller Kontostand für Konto " & _
    objHISAL("AuftraggeberKontoverbindung1", "Kontonummer1") & " bei BLZ " & _
    objHISAL("AuftraggeberKontoverbindung1", "Kreditinstitutcode1") & " ist " & _
    sSaldo
```

4.7.2 Beispiel: HBCI+

Da es mehrere Unterschiede zwischen HBCI und HBCI+ gibt, folgt ein weiteres Beispiel, diesmal auf Basis von HBCI+.

Nebenbei demonstriert dieses Beispiel auch eine andere Technik. Im Fall des HBCI+ werden ein Security-Objekt, ein Kunden-Objekt und ein Transport-Objekt erstellt, statt die entsprechenden vorgefertigten Objekte des DDBAC-Kontakts zu verwenden.

Diesmal ist der Code in Visual Basic verfasst, da die BACTransport-Komponente nicht scriptable ist.

```
Dim objBanking As BACBanking
Dim objCustomer As BACCustomer
Dim objTransport As IBACTransport
Dim objSecurity As BACSecurityTAN
Dim objDialog As BACDialog
Dim objHKSAL As BACSegment
Dim objHIUPD As BACSegment
Dim objHISAL As BACSegment
Dim objMessage As BACMessage
Dim sPIN As String
Dim sSaldo As String

' Zuerst wird das DDBAC Basis/Wurzel-Objekt BACBanking erstellt.
' Das BACBanking-Object ist ein "monostate"-Object, das heißt, dass es
' keinen local state besitzt. Alle Instanzen dieses Objektes sind identisch.
' Von dieser Basis kann auf die Collection BACContact zugegriffen werden. In diesem
' Beispiel greifen wir auf den allerersten Eintrag aus dieser Kollektion.
```

```
Set objBanking = CreateObject("DataDesign.BACBanking")

' Anstatt auf einen bereits konfigurierten Kontakt zurückzugreifen,
' wird hier ein HBCI+-Kundenobjekt erstellt.

Set objCustomer = objBanking.NewCustomer(280, "70000999", "2557")

' Um HBCI+ übers TCP/IP zu benutzen, erstellen wir das TCPIP-Transportobjekt.
' Falls Sie einen anderen Transporttyp benutzen (wie z.B. https), müssen Sie
' ein entsprechendes Objekt erstellen.
' Nach der Erstellung des Objekts, muss die korrekte IP-Adresse des Zielserver
' eingegeben werden. Gewöhnlich ist es ein https-Url wie
' https://hbcipus.bankname.com

Set objTransport = CreateObject("DataDesign.BACTransportTCPIP")
objTransport.CommunicationsAddress = "hbc102"

' Schließlich muss ein PIN/TAN Sicherheitsobjekt erstellen werden,
' das unmittelbar nach der Erstellung mit den Authentifizierungsdaten
' versorgt wird. Außer der PIN des Kontaktes werden beim HBCI+ keine
' weiteren Daten benötigt. Die PIN ist nur im virtuellen Sicherheitsmedium
' gespeichert und wird benutzt, wenn BeginDialog aufgerufen wird (siehe unten).

Set objSecurity = CreateObject("DataDesign.BACSecurityTAN")
Call objSecurity.Authenticate("DataDesign.BACSecurityTAN", "12345")

' Nun existieren alle nötigen Informationen um den HBCI+ Dialog zu beginnen.
' Das Einzige was noch getan werden muss, ist ein Dialogobjekt zu erstellen.

Set objDialog = objBanking.NewDialog(objCustomer, objTransport, objSecurity)

' Jetzt können wir die Kontodaten des Benutzers auslesen. Diese werden von
' der Bank während der Synchronisation übermittelt.

Call objDialog.BeginDialog(bacDialogSynchronizeCustomerSystemID)
Set objHIUPD = objCustomer.AccountData(0)
Call objDialog.EndDialog

' Nachdem die Synchronisation abgeschlossen wurde, eröffnen wir erneut die Verbindung
' mit dem Banksystem, doch diesmal wollen wir eine Standard HBCI-Transaktion ausführen.
' Dieser Teil ist sehr ähnlich mit dem des HBCI-Beispiels.

Call objDialog.BeginDialog(bacDialogStandard)

' Erstellt ein HKSAL (Saldo) Auftragssegment. Für dieses Beispiel greifen
' wir auf das allererste Konto aus den für den Kunden verfügbaren Konten.
' Für jedes Konto ist ein HIUPD-Segment in der Segment-Collection des
' AccountData-Objektes gespeichert.

Set objHKSAL = objBanking.NewSegment("HKSAL", 3)
objHKSAL("AuftraggeberKontoverbindung1", "Kontonummer1") = _
    objHIUPD("Kontoverbindung1", "Kontonummer1")
objHKSAL("AuftraggeberKontoverbindung1", "Laenderkennzeichen1") = _
    objHIUPD("Kontoverbindung1", "Laenderkennzeichen1")
```

```

objHKSAL("AuftraggeberKontoverbindung1", "Kreditinstitutcode1") = _
    objHIUPD("Kontoverbindung1", "Kreditinstitutcode1")
objHKSAL("AlleKonten1") = False
objHKSAL("Kontowaehrung1") = objHIUPD("Kontowaehrung1")

' Hier gibt es den größten Unterschied zwischen der HBCI+ Kommunikation
' und HBCI. Wir müssen eine TAN eingeben, um die Nachricht, die wir versenden
' wollen, zu signieren. Falls keine TAN in die Parameterkollektion
' aufgenommen wird, wird die DDBAC dem Benutzer ein TAN Eingabe-Dialog angezeigt.
' Der angegebene Wert wird nach der Ausführung der Transaktion gelöscht, daher
' muss bei jeder einzelnen Transaktion eine TAN angegeben werden.

objSecurity.Parameter("TAN") = "12345"

' Ausführung der Transaktion und Beendigung des Dialogs.
' Die Antwort wird im BACMessage-Objekt aufgefangen.
Set objMessage = objDialog.ExecuteSegment(objHKSAL)
objDialog.EndDialog Nothing

' Auswertung der Antwort. Wir überprüfen hier keine Fehler, sondern
' greifen auf das allererste Segment aus der ResponseSegments-Kollektion.
Set objHISAL = objMessage.Transactions(0).ResponseSegments(0)

sSaldo = objHISAL("SaldoGebucht1", "Waehrung1") & " "
If objHISAL("SaldoGebucht1", "SollHabenKennzeichen1") = "D" Then
    sSaldo = sSaldo & "-"
End If
sSaldo = sSaldo & objHISAL("SaldoGebucht1", "Wert1")

MsgBox "Aktueller Kontostand für Konto " & _
    objHISAL("AuftraggeberKontoverbindung1", "Kontonummer1") & " bei BLZ " & _
    objHISAL("AuftraggeberKontoverbindung1", "Kreditinstitutcode1") & " ist " & _
    sSaldo

```

4.7.3 Zusammenfassung

Dieser Abschnitt zeigt detaillierte Informationen über die einzelnen DDBAC-Objekte und deren Verwendung. Zusätzlich gibt es ein komplettes Referenzhandbuch, das alle Interfaces, Objekte, Methoden und Konstanten dokumentiert. Das Handbuch ist unter „DDBAC Reference Manual.html“ verfügbar.

4.8 Arbeit mit den DDBAC-Objekten

4.8.1 Das BACSegment-Objekt

Das BACSegment ist das Arbeitspferd der DDBAC Komponenten. Es ist verantwortlich für die Speicherung aller zu einem HBCI Segment gehörigen Daten, sowie der Darstellung im HBCI Format. Ein HBCI Segment ist eine Folge von Datenelementgruppen (DEG) die wiederum Gruppenelemente (GD) enthalten. Beachten Sie bitte, dass dies eine Vereinfachung der offiziellen HBCI Terminologie ist. Die Vereinfachung nimmt an, dass ein atomares HBCI Datenelement ein spezieller Fall der DEG ist, der nur ein einzelnes Element (GD) enthält.

Die einzelnen Gruppenelemente eines Segmentes können durch die Position des DEG innerhalb des Segmentes und die Position des GD innerhalb der Datenelementgruppe adressiert werden (siehe auch HBCI 2.0 Abschnitt II.8.5.2). Zusätzlich zu einer Adressierung über ihre Position, kann durch das BACSegment ein GD auch über einen eindeutigen Namen angesprochen werden. Diese flexible Adressierung wird durch die definition von HBCI Syntax Deskriptoren erreicht. Ein HBCI Syntax Deskriptor beschreibt den vollständigen Aufbau eines HBCI Segmentes und legt für die einzelnen Elemente Namen fest. Anhand des Syntax Deskriptors kann das BACSegment Objekt ein HBCI Segment generieren und interpretieren.

Die DDBAC enthalten bereits vordefinierte HBCI Syntax Deskriptoren für alle in der HBCI Version 2.0.1 definierten Segmenttypen. Eine Anwendung kann bei Bedarf jedoch jederzeit weitere Segmente definieren und nutzen.

Es ist sichergestellt, dass jede BACSegment-Instanz als ein einzelnes HBCI Nachrichtensegment konvertiert und gespeichert werden kann. Dieses Nachrichtensegment wird, gemäß den HBCI Segment-Syntax-Regeln, als ein String repräsentiert.

Um eine Segmentinstanz aus einem und in ein HBCI-Segment zu konvertieren, muss die Syntax angegeben sein, die das gewünschte Segmentlayout beschreibt. Das BAC beinhaltet eine komplette Kollektion vordefinierter Syntaxbeschreibungen für alle bekannten HBCI-Segmente. Mehr Informationen über die Syntaxbeschreibung finden Sie im Kapitel 5.1 (Das neue HBCI-Syntaxobjektmodell).

4.8.1.1 Datenelemente-Container

Der HBCI Segment Generator baut ein HBCI Segment anhand eines gegebenen Syntax Deskriptors auf. Die eingestellten Daten werden dabei aus dem Datenelement Container des BACSegment Objektes entnommen. Die gewünschten Elemente werden im Container über die Position und ihren Namen gewählt.

Umgekehrt verhält es sich bei der Zerlegung eines HBCI Segmentes. Die aus dem Segment extrahierten Daten werden in den Datenelement Container zusammen mit ihrer Position und dem Namen aus dem Syntax Deskriptor abgelegt.

Die Indizierung in den assoziativen Datenelement Container steht in engem Zusammenhang mit dem HBCI Syntax Deskriptor.

Der Datenelement Container ist eine assoziative Tabelle, die eine Kombination aus vier Indizes auf einen Datenelementwert abbildet. Die vier Indizes werden wie folgt definiert:

Typ	Name	Index Beschreibung
String	DEGName	Lesbarer Name der Datenelementgruppe (DEG) in der dieses Element enthalten ist. Falls das Element ein einzelnes DE ist, so ist dies der Name dieses Datenelementes. Dieser Name muss für das DEG innerhalb des Segmentes eindeutig sein. Dieser Index kann auch null sein, z.B. wenn durch den Syntax Descriptor kein Name vergeben wurde. Der Name sollte nur einfache ASCII Zeichen enthalten und darf auf keinem Fall mit einer Ziffer enden.
Long	DEGPos	Position der DEG oder des DE im HBCI Segment. Die Nummerierung entspricht den HBCI Konventionen (HBCI II.8.5.2 Nr. 2). Das heißt, der Segmentkopf hat die DEG Position eins. Der Positionswert null wird verwendet um anzuzeigen, dass die tatsächliche Position nicht bekannt ist. Negative Werte sind illegal.
String	GDName	Lesbarer Name des Gruppendatenelementes (GD). Ist das indizierte Element ein einfaches DE, so ist dies der Null-String. Dieser Name muss das GD innerhalb seiner DEG eindeutig bezeichnen. Dieser Index kann auch Null sein, wenn z.B. kein Name vergeben wurde.
Long	GDPoS	Position des GD innerhalb seiner DEG. Die Nummerierung entspricht den HBCI Konventionen (HBCI II.8.5.2 Nr. 2). D.h. das erste GD hat die Position eins. Der Positionswert null bedeutet, dass die tatsächliche Position unbekannt ist. Negative Werte sind ungültig.

Eine gültige Indexkombination erfordert, dass zumindest einer von DEGName und DEGPos, und zumindest einer von GDName und GDPoS gegeben sind (d.h. sind nicht null).

Um die Abbildung auf mehrfach wiederholte Datenelemente zu ermöglichen, wird dem Namen immer der Wiederholungszähler angehängt. Zum Beispiel, die DEG „Rueckmeldung“ darf in einem „HIRMG“ Segment bis zu 99-mal wiederholt werden. Das erste auftauchende DEG hätte demnach den Namen „Rueckmeldung1“, das zweite wäre „Rückmeldung2“, und so weiter. Falls ein Element genau einmal auftreten darf muss dennoch die Ziffer „1“ angehängt werden. So hat zum Beispiel die DEG „Kreditinstitutskennung“ immer den Namen „Kreditinstitutskennung1“. Das alles gilt für DEG Namen genauso wie für GD Namen.

Wenn ein Segment zerlegt wird, werden die extrahierten Elemente immer mit allen vier Indizes in den Datenelement Container abgelegt. Zum Beispiel würden beim Zerlegen des oben gezeigten „HKUEB“ Segmentes die folgenden Werte im Datenelement Container abgelegt:

DEGName	DEGPos	GDName	GD Pos	Wert
„AuftraggeberKontoverbindung1“	2	„Kontonummer1“	1	„1234567“
„AuftraggeberKontoverbindung1“	2	„Laenderkennzeichen1“	2	„280“
„AuftraggeberKontoverbindung1“	2	„Kreditinstitutcode1“	3	„10020030“
„EmpfaengerKontoverbindung1“	3	„Kontonummer1“	1	„7654321“
„EmpfaengerKontoverbindung1“	3	„Laenderkennzeichen1“	2	„280“
„EmpfaengerKontoverbindung1“	3	„Kreditinstitutcode1“	3	„20030040“
„EmpfaengerNameEins1“	4	"" (Null)	1	„MEIER FRANZ“
„Zahlungsbetrag1“	6	„Wert1“	1	„1000,“
„Zahlungsbetrag1“	6	„Waehrung1“	2	„EUR“
„Textschluessel1“	7	"" (Null)	1	„51“
„Textschluesselergaenzung1“	8	"" (Null)	1	„000“
„Verwendungszweck1“	9	„Zeile1“	1	„RE-NR.1234“
„Verwendungszweck1“	9	„Zeile2“	2	„KD-NR.9876“

Wie man sieht, wird ein einzelnes DE lediglich als Sonderfall einer DEG mit nur einem GD ohne Namen behandelt. Dieses GD hat immer die Position eins.

Falls die Anwendung nun den Zahlungsbetrag auslesen möchte, kann es dies tun indem sie eine der folgenden Indexkombinationen angibt:

DEGName	DEGPos	GDName	GDPos	Bemerkung
"Zahlungsbetrag1"	0	"Wert1"	0	DEG und GD werden durch den Namen adressiert.
"Zahlungsbetrag1"	0	" "	1	Die DEG wird durch ihren Namen adressiert, das GD durch ihre Position innerhalb des DEG.
" "	6	"Wert1"	0	Die DEG wird durch ihre Position innerhalb des Segmentes adressiert, das GD durch seinen Namen.
" "	6	" "	1	DEG und GD werden durch ihre jeweiligen Positionen adressiert.

Um das einzelne DE "EmpfängerNameEins1" auszulesen, kann die Anwendung eine der folgenden Indexkombinationen verwenden:

DEGName	DEGPos	GDName	GDPos	Bemerkung
"EmpfängerNameEins1"	0	" " (Null)	1	Das DE wird durch seinen Namen angesprochen.
" "	4	" " (Null)	1	Das DE wird durch seine Position innerhalb des Segmentes angesprochen.

Wenn ein Segment generiert werden soll, muss die Anwendung erst den Datenelement Container mit den gewünschten Werten füllen. Die Anwendung kann die Werte für die einzelnen Datenelemente mit Hilfe beliebiger gültiger Indexkombinationen festlegen. Um das Beispiel von oben fortzuführen, das Betragesfeld ("Zahlungsbetrag1", "Wert1") des HKUEB Segmentes kann mit folgenden gültigen Indexkombinationen belegt werden:

DEGName	DEGPos	GDName	GDPos	Bemerkung
"Zahlungsbetrag1"	0	"Wert1"	0	DEG und GD werden durch den Namen adressiert.
"Zahlungsbetrag1"	0	" "	1	Die DEG wird durch ihren Namen adressiert, das GD durch ihre Position innerhalb des DEG.
" "	6	"Wert1"	0	Die DEG wird durch ihre Position innerhalb des Segmentes adressiert, das GD durch seinen Namen.
" "	6	" "	1	DEG und GD werden durch ihre jeweiligen Positionen adressiert.

Normalerweise wird der Anwendungsprogrammierer die Indizierung mit Hilfe der definierten Namen vorziehen. Bleibt die Frage, woher der Anwendungsprogrammierer die Namen der einzelnen DEGs und GDs erfährt?

Die Antwort liegt in der Syntaxbeschreibung. Einer der Zwecke der Syntaxbeschreibung ist, die Namen und die Positionen eines jeden Datenelementes in einem Segment zu identifizieren. Weitere Informationen über die Syntaxbeschreibung siehe Kapitel 5.1 (Das neue HBCI-Syntaxobjektmodell).

4.8.2 Das BACCustomer-Objekt

Das BACCustomer-Objekt repräsentiert einen HBCI-Kontakt; daher sollte es besser BACCcontact genannt werden. Der Name BACCcustomer wird jedoch aus Kompatibilitätsgründen benutzt. BACCcustomer repräsentiert und verwaltet alle Daten eines einzelnen HBCI-Kontaktes. Ein HBCI-Kontakt wird einmalig identifiziert durch die Kombination des Landescodes, des Bankcodes und der BenutzerID.

4.8.2.1 Die HBCI-Kontaktentabelle

Der Benutzer erstellt einen neuen HBCI-Kontakt durch den HBCI-Kontakte-Administrator (Systemsteuerung). Alle HBCI-Kontakte eines Systems sind in einer globalen Datei HBCI-Kontaktentabelle gespeichert. Diese Tabelle enthält eine Sammlung mit Namen- und Wertenattributen (Hash-Values) für jeden Kontakt. Alle Attribute in dieser Tabelle sind über die property-Field des BACCcustomer-Objektes zugänglich.

Ein Datensatz der HBCI-Kontaktentabelle wird eindeutig identifiziert durch die Kombination des Landescodes (280 in Deutschland), der Bankleitzahl und der Benutzerkennung. Der Datensatz ist unabhängig von der KundenID.

Benutzerkennung versus KundenID

Der Zweck der HBCI "Benutzerkennung" (die in der DDBAC durchgängig zur "UserID" übersetzt wurde) ist es, eine reale Person innerhalb eines Kreditinstituts eindeutig zu identifizieren. Dadurch überträgt eine Bank eine einmalige HBCI "Benutzerkennung" auf eine Person, wenn sie den HBCI-Zugang einrichtet. Natürlich können verschiedene Banken diverse HBCI "Benutzerkennungen" für eine Person festlegen; die HBCI "Benutzerkennung" ist deshalb nur innerhalb des Kontextes einer einzelnen Bank einmalig. Die HBCI "Benutzerkennung" erscheint nur in den von dieser Person erstellten digitalen Signaturen, mit der sie sich selbst gegenüber der Bank identifiziert.

Im Gegensatz dazu ist der Zweck der HBCI "KundenID" (die in der DDBAC durchgängig zur "CustomerID" übersetzt wurde) eine Rolle zu identifizieren, die ein Kunde einnimmt. Dadurch kann die Bank Einzelpersonen, Ehepaare, Firmen, etc. identifizieren. Auch hier ist eine "KundenID" nur innerhalb des Kontextes einer einzelnen Bank einmalig. Verschiedene Banken können diverse KundenID für unterschiedliche Rollen festlegen. Eine "KundenID" wird von der Bank gewöhnlich festgelegt, wenn eine Rechtsperson mit der Bank eine Geschäftsbeziehung eingeht, zum Beispiel, beim Öffnen eines Girokontos. Die HBCI "KundenID" wird der Bank vom Benutzer in einer Dialoginitialisierungsnachricht übermittelt, die die Rechtsperson für den Verlauf dieses Onlinedialogs identifiziert.

Da eine Einzelperson mehrere, unterschiedliche Rollen einnehmen kann, muss sie die Möglichkeit haben, noch vor dem Beginn eines Onlinedialogs eine der möglichen unterschiedlichen "KundenIDs" auszuwählen. Während des Onlinedialogs kann nur auf Konten, die zu dieser Rolle gehören, zugegriffen werden. In HBCI wird die Assoziation der "Benutzerkennung" und der "Kunden-ID" eine Rolle genannt.

Wenn sich eine Person erstmals bei einer Bank anmeldet, erhält sie eine Liste aller Konten, auf die sie zugreifen kann. Diese Liste (genannt UPD, verfügbar durch das BACAccountData-Objekt in der DDBAC) beinhaltet die "KundenID" der Rolle, die jedes Konto "besitzt". Dadurch ist es möglich eine Liste aller möglichen "KundenIDs" für eine Person zu erstellen, sobald ihre UPD gefunden wurde. Selbst bei der ersten Anmeldung ist es erforderlich, eine KundenID mitzusenden. Aus diesem Grund fordert die DDBAC während der Einrichtung eines neuen Homebanking Kontakts eine einzelne Standard-"Kunden-ID".

In vielen Fällen entspricht die "Benutzerkennung" der "KundenID". Dies wird aus Gründen der Vereinfachung gemacht für den Fall, wenn eine Person gleichzeitig auch ihre eigene Rolle einnimmt. Aus diesem Grund nimmt die DDBAC den Wert der "Benutzerkennung" als Standardwert für die "Kunden-ID" an, falls keine "Kunden-ID" für einen HBCI "Homebanking Kontakt" eingegeben wurde.

Die HBCI-Kontaktentabelle ist in der Datei DDUSERS.DAT gespeichert. Der Speicherort kann durch den Registry-Key "DataDir" festgelegt werden (siehe 2.2 Registry-Einträge).

Eine Applikation kann die HBCI-Kontaktentabelle nutzen um zusätzliche Attribute abzuspeichern (sie Property Field des BACCustomer-Objekts). Applikationsspezifische Attribute sollten eindeutige Namen der Bezeichner verwenden, z.B. durch Prefixing mit dem Applikationsnamen.

4.8.2.2 Aufbewahrung der UPD- und BPD-Dateien

HBCI definiert die beiden Parametersätze UPD (User Parameter Data) und BPD (Bank Parameter Data). Jeder Satz ist in seiner eigenen Datei im Datenverzeichnis gespeichert. Die UPD-Dateien verwenden den Dateinamensuffix ".upd", die BPD-Dateien ".bpd". Der Dateiname als solcher wird als eindeutige Kombination des Ländercodes, der Bankleitzahl und der Benutzerkennung erstellt. Zum Beispiel:

```
280_60010111_10100460.upd
280_10010111_63520101.upd
280_60010111.bpd
280_10010111.bpd
```

4.8.3 Das BACDialog-Objekt

Das BACDialog Objekt ist verantwortlich für die Steuerung des HBCI Online Dialoges und den damit verbundenen Austausch von HBCI Nachrichten über die Transportschnittstelle. Ein HBCI

Dialogablauf beginnt mit einer Dialoginitialisierung, danach folgen die Auftragsnachrichten. Zuletzt wird der Dialog ordnungsgemäß mit einer HBCI Dialogbeendigung abgeschlossen. Das BACDialog Objekt steuert genau diesen Dialogablauf.

4.8.3.1 Erstellen eines BACDialog-Objektes

Ein BACDialog Objekt kann nur durch die Methode BACCreateDialog des BACBanking Objektes erstellt werden. Durch diese Methode werden das BACCustomer Objekt sowie die Kunden-ID für die gesamte Lebensdauer des BACDialog Objekts festgelegt. Ist BACCreateDialog erfolgreich gibt es eine Referenz auf das neu erstellte BACDialog Objekt zurück.

4.8.3.2 HBCI-Dialogoptionen

Das BACDialog Objekt hat verschiedene Eigenschaften die Optionen für den HBCI Dialog festlegen. Wurde das BACDialog Objekt frisch kreiert, sind diese Eigenschaften mit sinnvollen Vorgabewerten belegt. Ein Anwendungsprogramm sollte von der Vorgabe abweichende Optionen sofort nach dem Erstellen des BACDialog Objektes setzen. Folgende Tabelle gibt einen Überblick über die vorhandenen Optionen.

Optionseigenschaft	Datentyp	Default	Beschreibung
Sprache	BACDialog-Sprachen	bacDialog-LanguageDefault	Diese Option wählt die HBCI Dialogsprache. Der Vorgabewert wählt die Standardsprache des Banksystems. Zur Auswahl stehen die Sprachen Deutsch (bacDialogLanguageGerman), Englisch (bacDialogLanguageEnglish) und Französisch (bacDialogLanguageFrench).
HBCIVersion	Long	bacVersionHBCI	Diese Option legt die Version der HBCI Spezifikation fest, nach der der Dialog geführt werden soll. Der Wert dieser Eigenschaft ist um 100 skaliert, d.h. die HBCI Version 2.0.1 wird als 201 angegeben.
AutoSignature	Boolean	True	Diese Eigenschaft legt fest, ob alle abgehenden HBCI Nachrichten durch das Security Objekt signiert werden sollen (True), oder nicht (False).
DialogType	BACDialogTypes	bacDialogStandard	Diese Eigenschaft legt fest, ob alle abgehenden HBCI Nachrichten durch das Security Objekt verschlüsselt werden sollen (True), oder nicht (False).

4.8.3.3 Dialogzustand

Um den aktuellen Dialogzustand korrekt abbilden zu können verwendet das BACDialog Objekt verschiedene Eigenschaften. Folgende Tabelle zeigt die Eigenschaften die gemeinsam den "Zustand" des BACDialog Objektes definieren.

Optionseigenschaft	Datentyp	Beschreibung
DialogID	String	Diese ID identifiziert den Dialog eindeutig und wird vom Banksystem während einer Dialoginitialisierung festgelegt.
State	BACDialogStates	Gibt den expliziten Zustand des BACDialog Objektes und somit des HBCI Dialoges wieder. Die möglichen expliziten Dialogzustände und Ihre Übergänge sind unten erklärt.
NextMessageSeqNo	Long	Wird zur Nummerierung der abgehenden HBCI Nachrichten verwendet. Die erste Nachricht eines Dialoges hat die Nummer 1.

Ein neues BACDialog Objekt beginnt immer mit dem expliziten Dialogzustand bacDialogIdle. Im Laufe eines Dialoges durchwandert er verschiedene Zustände um nach Beendigung des Dialoges wieder in den ursprünglichen Zustand bacDialogIdle zurückzukehren. Das folgende Diagramm gibt einen Überblick über die Abfolge der Dialogzustände.

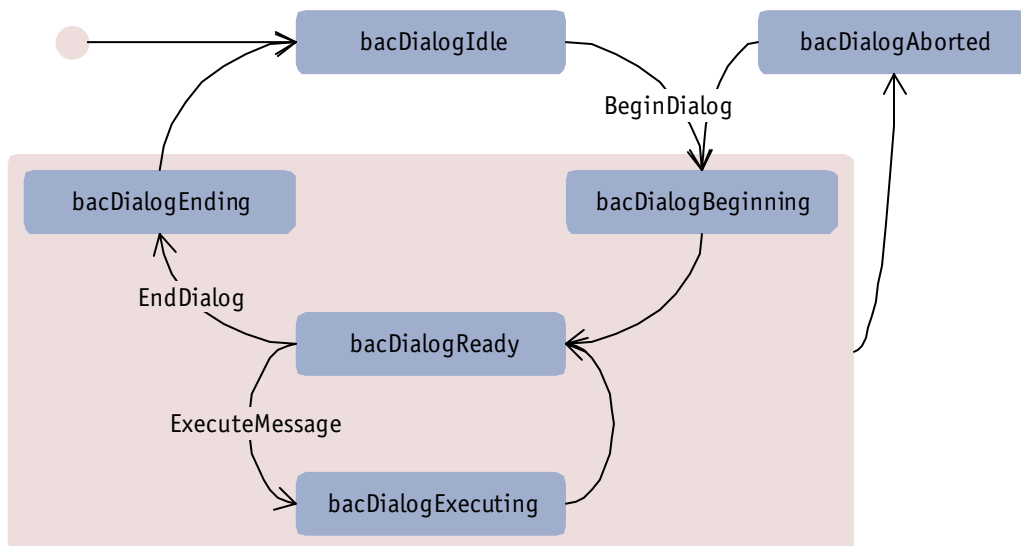


Abb. 3 - BACDialog Zustandsdiagramm

Die gezeigten Zustände `bacDialogBeginning`, `bacDialogExecuting` und `bacDialogEnding` sind instabile Zustände die nur für die Dauer der Bearbeitung der initiierten Operation angenommen werden. Wurde die Operation abgeschlossen wird automatisch in den nächsten stabilen Zustand weitergeschaltet. Tritt während der Abarbeitung ein Fehler auf wird der Dialog abgebrochen und in den Zustand `bacDialogAborted` geschaltet.

Zustand	Beschreibung
bacDialogIdle	In diesem Zustand kann ein neuer Dialog durch Aufrufen der Methode BeginDialog begonnen werden. Das BACDialog Objekt geht daraufhin in den Zustand bacDialogBeginning über.
bacDialogBeginning	Eine Dialoginitialisierung wird gerade durchgeführt. Am Ende einer erfolgreichen Dialoginitialisierung steht die DialogID zur Verfügung und das BACDialog Objekt schaltet in den Zustand bacDialogReady.
bacDialogReady	Nur in diesem Zustand können Aufträge an das Banksystem mittels ExecuteMessage aufgegeben werden. Wird ein Auftrag initiiert, schaltet das BACDialog Objekt in den Zustand bacDialogExecuting.
bacDialogExecuting	Eine Auftragsnachricht wird gerade bearbeitet. Nachdem die Antwortnachricht eingetroffen ist, werden NextMessageSeqNo und NextBankMessageSeqNo um einen Zähler hoch gezählt und das BACDialog Objekt geht wieder in den Zustand bacDialogReady über.
bacDialogEnding	Eine ordnungsgemäße Dialogbeendigung wird durchgeführt. Nachdem der Dialog vollständig beendet und die Transportverbindung getrennt wurde, wird in den Zustand bacDialogIdle geschaltet.
bacDialogAborted	Immer wenn bei der Abarbeitung einer Operation ein Fehler auftritt wird der Dialog abgebrochen, die Transportverbindung getrennt und das BACDialog Objekt geht in den Zustand bacDialogAborted über. In diesem Zustand kann ein neuer Dialog begonnen werden.

4.8.3.4 Synchron und asynchrone Operationen

Das BACDialog Objekt hat genau drei Methoden die tatsächlich eine Online-Operation durchführen: BeginDialog, ExecuteMessage und EndDialog. Jede dieser Methoden kann ihre Operation sowohl synchron als auch asynchron durchführen. Wird die Operation synchron durchgeführt kehrt der Methodenaufruf erst zurück, nachdem die Operation erfolgreich beendet wurde, oder ein Fehler den Abbruch der Operation erzwungen hat. Wird die Operation asynchron durchgeführt, kehrt der Methodenaufruf sofort zurück und die eigentliche Operation wird nebenläufig durchgeführt.

Bei einer synchronen Ausführung kehrt die Methode immer mit einem stabilen Zustand des BACDialog Objektes zurück. Zum Beispiel, wird im Zustand bacDialogIdle die Methode BeginDialog aufgerufen, kehrt sie entweder im Zustand bacDialogReady oder bacDialogAborted zurück. Der Zustand bacDialogBeginning wird nur für die Dauer des Methodenaufrufes angenommen.

Bei einer asynchronen Ausführung hingegen, kehrt die Methode mit einem instabilen Zustand des BACDialog Objektes zurück. Wird also BeginDialog asynchron verwendet, so kehrt der Methodenaufruf mit dem BACDialog Zustand bacDialogBeginning zurück. Ob eine Operation

synchron oder asynchron ausgeführt wird, hängt vom Wert des Parameters "Timeout" beim Aufruf der Methode ab. Wird als "Timeout" null übergeben, so wird die Operation asynchron und ohne Timeout durchgeführt. Wird ein von null verschiedener Wert übergeben, so wird die Operation synchron mit der gegebenen Zeitbegrenzung ausgeführt.

4.8.4 Behandlung von Fehlern und Ausnahmen

Normalerweise werden alle auftretenden Fehler über den Standard COM Automation Mechanismus zur Ausnahmeverarbeitung gemeldet. Das heißt, das Anwendungsprogramm kann die gewohnten Ausnahmeverarbeitungsmechanismen seiner Entwicklungsumgebung nutzen³. Im Folgenden wird bei derart gemeldeten Fehlern immer von synchronen Ausnahmen gesprochen.

Zusätzlich implementiert das Objekt BACDialog einen asynchronen Mechanismus zur Fehleranzeige. Dieser ist unten im Detail erläutert.

Die synchron ausgelösten Ausnahmen lassen sich ungefähr in zwei Kategorien einteilen: 1) System- und Programmfehler, und 2) DDBAC spezifische Fehler.

4.8.4.1 Synchroner System- und Programmfehler

Systemausnahmen werden durch fatales Systemversagen, wie z.B. Speicherplatzmangel meist direkt durch das Betriebssystem ausgelöst. Hinzu kommen Programmfehler, d.h. Fehler die Ihre Ursache in einem Programmierfehler in den DDBAC oder dem Anwendungsprogramm selbst haben, z.B. ein Methodenaufruf mit illegalen Parametern. In beiden Fällen generieren die DDBAC eine durch das Betriebssystem definierte allgemeine Systemausnahme⁴.

4.8.4.2 Das COM Automation Fehlerobjekt

Für die meisten synchron gemeldeten Fehlerausnahmen wird von den DDBAC ein beschreibendes COM Automation Fehlerobjekt erzeugt. In C++ kann dieses Fehlerobjekt durch die Win32 API Funktion GetErrorInfo() ermittelt werden. In Visual Basic liegt dieses Fehlerobjekt durch das global definierte Err Objekt vor.

Allgemein wird das Fehlerobjekt wie folgt initialisiert:

³ Für C++ bedeutet das, daß ein fehleranzeigender HRESULT Wert und teilweise auch ein ErrorInfo Objekt generiert und gesetzt wird. Das ErrorInfo Objekt kann über die Win32 API Funktion GetErrorInfo() abgeholt werden. Siehe dazu auch das "Microsoft Automation Reference Manual".

Für Visual Basic werden für alle Fehler Ausnahmen generiert die mit "On Error GoTo" behandelt werden können. Das Fehlerobjekt liegt in der globalen Variablen "Err" vor.

⁴ Das heißt, der Methodenaufruf liefert einen allgemeinen fehleranzeigenden HRESULT Wert ohne daß dafür ein Fehlerobjekt generiert würde.

VB: Err Object	C++: IErrorInfo	Beschreibung
Err.Number	HRESULT returned from method	Die Fehlernummer. Für DDBAC spezifische Fehler (siehe unten) ist der „Err.Number“ Wert im Bereich Null bis 10999, das entsprechende HRESULT aber im Bereich 800A0000 ₁₆ bis 800A2AF7 ₁₆ . Ansonsten sind die Werte identisch.
Err.Source	IErrorInfo::GetSource()	Hier wird die sog. ProgID des Objektes eingestellt, welches das Fehlerobjekt generiert, z.B. „DataDesign.BACSegment“.
Err.Description	IErrorInfo::GetDescription()	Hier wird eine Beschreibung des Fehlers eingetragen.
Err.HelpFile	IErrorInfo::GetHelpFile()	Derzeit nicht benutzt.
Err.HelpContext	IErrorInfo::GetHelpContext()	Derzeit nicht benutzt.
	IErrorInfo::GetGUID()	Derzeit nicht benutzt.

4.8.4.3 DDBAC-spezifische Fehlercodes

Zusätzlich zu den Systemfehlern gibt es noch eine Reihe spezieller Fehlercodes die allein durch die DDBAC definiert sind. Derartige Fehler können sowohl synchron als HRESULT als auch asynchron durch das BACDialog Objekt angezeigt werden.

Durch die DDBAC definierte Fehlercodes liegen im Wertebereich 10000 bis 10999. Der Wertebereich Null bis 9999 ist für die Anzeige asynchroner HBCI Rückmeldungs-codes reserviert, so dass sich ein gesamter Wertebereich von Null bis 10999 ergibt.

Werden DDBAC spezifische Fehlercodes als HRESULT dargestellt, so wird zum eigentlichen Fehlercode 800A0000₁₆ addiert, d.h. der HRESULT Wert liegt im Bereich 800A0000₁₆ bis 800A2AF7₁₆. Im Gegensatz zu anderen HRESULT Werten, wird dieser Bereich in der Number Eigenschaft des Err Objektes von Visual Basic als Null bis 10999 angezeigt.

Derzeit sind die folgenden DDBAC spezifischen Fehlercodes definiert:

Symbolname	Fehlercode	Beschreibung
bacErrorCantBuildMessage	10000	Die gewünschte HBCI Nachricht konnte nicht generiert werden. Die Ursache liegt dabei meist darin, daß ein benötigtes Datenelement fehlt oder falsch belegt wurde. Eine genaue Fehlerbeschreibung wird als HBCI Rückmeldung (siehe unten) eingestellt.
bacErrorTransportFailure	10001	Fehler bei der Datenübertragung. Der Dialog wurde abgebrochen.
bacErrorCantParseMessage	10002	Die vom Kreditinstitut empfangene Nachricht konnte nicht dekodiert werden. Eine genaue Fehlerbeschreibung wird als HBCI Rückmeldung eingestellt.
bacErrorDialogAborted	10003	Der Dialog wurde durch den Benutzer (Aufruf der Methode BACDialog.Abort) abgebrochen.
bacErrorTimeout	10004	Es wurde keine Antwortnachricht innerhalb der gesetzten maximalen Wartezeit empfangen. Der Dialog wurde daraufhin abgebrochen.
bacErrorSecurity	10005	Die gewünschte HBCI Nachricht konnte nicht generiert werden. Die Ursache liegt dabei meist darin, daß ein benötigtes Datenelement fehlt oder falsch belegt wurde. Eine genaue Fehlerbeschreibung wird als HBCI Rückmeldung (siehe unten) eingestellt.

4.8.4.4 HBCI-Rückmeldungen

Im HBCI Protokoll werden Fehler (und Erfolg) durch sogenannte Rückmeldungen angezeigt (siehe HBCI 2.0.1 Abschnitt II.8.5). Eine HBCI Rückmeldung enthält unter anderem einen numerischen Rückmeldungscode und einen Rückmeldungstext. Die HBCI Spezifikation erwartet daß alle Kundenprodukte zumindest den in der Rückmeldung enthaltenen Klartext dem Kunden anzeigen (siehe HBCI 2.0.1 II.8.5.1). Hier einige Beispiele für mögliche HBCI Rückmeldungen:

“9330:Elektronische Signatur gesperrt”

“9210:15:Kontonummer existiert nicht”

“3020:3,5:Bankleitzahl veraltet, die neue BLZ lautet 10020030:10020030”

Das BACDialog Objekt kann HBCI Rückmeldungen als normale Fehler anzeigen. Dazu wird der Rückmeldungscode als Fehlercode und die gesamte Rückmeldung als Fehlerbeschreibung in das Fehlerobjekt eingestellt. Das BACDialog Objekt wertet aber immer nur die Rückmeldung auf die Gesamtnachricht (HBCI Segment HIRMG) aus. Die Rückmeldungen auf die einzelnen Aufträge bleiben unberücksichtigt. Außerdem werden nur HBCI Rückmeldungscode der Fehlerklasse neun, d.h. im Bereich von 9000 bis 9999 als Fehler angezeigt. Andere Rückmeldungscode werden als Erfolg und nicht als Fehler bewertet.

In jedem Fall aber, werden der Rückmeldungscode und die Rückmeldung in die BACDialog Eigenschaften HBCIErrorCode und HBCIErrorDescription eingestellt.

4.8.4.5 Asynchrone Fehler

Das BACDialog Objekt erledigt praktisch alle Aufgaben nebenläufig und kann auftretende Fehler deshalb oft nicht durch eine COM Automation Ausnahme anzeigen. Nichtsdestotrotz werden alle Fehler die sofort beim Methodenaufruf diagnostiziert werden können auch als synchrone

Ausnahmen angezeigt. Die asynchrone Fehleranzeige greift erst, nachdem eine Methode mit Erfolg zum Aufrufer zurückgekehrt ist und eine nebenläufige Aktivität gestartet wurde.

Alle nebenläufigen Aktivitäten des BACDialog Objektes führen letztendlich zu einer erfolgreichen oder nicht erfolgreichen Abarbeitung. Das Ende der Aktivität wird über das Ereignis BACDialog.OnActivityComplete angezeigt. Diesem Ereignis wird ein Fehlercode mitgegeben mit dem das Anwendungsprogramm feststellen kann ob die Aktivität erfolgreich war, oder nicht. Dieser Fehlercode ist außerdem in die Eigenschaft BACDialog.HBCIErrorCode eingestellt. Darüber hinaus können im Fehlerfall zusätzliche Informationen in den Eigenschaften BACDialog.HBCIErrorDescription und BACDialog.TransportError vorliegen.

Bei Aktivitäten die einen tatsächlichen Dialog mit dem Banksystem führen, wird jeweils nur das Rückmeldesegment „HIRMG“ der Antwortnachricht ausgewertet⁵. Die auftragspezifischen Rückmeldungen „HIRMS“ werden nicht ausgewertet und deshalb nie als Fehler gemeldet! Bei der Auswertung des „HIRMG“ Segmentes werden die RückmeldungsCodes aller enthaltenen Rückmeldungen geprüft. Die Rückmeldung mit dem numerisch höchsten Rückmeldungscode wird dann in die Eigenschaften BACDialog.HBCIErrorCode und BACDialog.HBCIErrorDescription eingestellt und gemeldet. Dabei wird in die Eigenschaft BACDialog.HBCIErrorDescription die gesamte Rückmeldung eingestellt.

Fehler die durch das Transportobjekt verursacht werden, führen immer zum Abbruch des HBCI Dialoges und werden durch den Fehlercode „bacErrorTransportFailure“ angezeigt. In diesem Fall kann eine genauere Fehlerursache durch die Eigenschaft BACDialog.TransportError ermittelt werden.

4.9 SEPA-Erweiterungen

Seit Version 4.2.0 beherrscht die DDBAC auch SEPA-Aufträge. SEPA (Single Euro Payment Area) wird zum 28.01.2008 in Deutschland und im EU-Raum eingeführt werden und soll mittelfristig den bestehenden EU-weiten Zahlungsverkehr ablösen.

4.9.1 Geschäftsvorfälle

Für SEPA wurden in „FinTS 3.0 SEPA Erweiterung“ viele neue Geschäftsvorfälle spezifiziert. Die Syntax dieser Geschäftsvorfälle ist in der HTML Seite „DDBAC SEPA Segmente.html“ enthalten. Diese wird beim SDK mitgeliefert.

⁵ Dieses Segment liegt außerdem in der Eigenschaft Acknowledgement des BACMessage Objektes vor.

Bei bereits bekannten Geschäftsvorfällen wie „Kontoumsätze anfordern“ (HKKAZ/HKKAN/HKEKA/HKKAU) wurde lediglich die Segmentversion erhöht. Im Wesentlichen hat sich bei all diesen Geschäftsvorfällen das DEG „Kontoverbindung“ verändert. Das Feld Kontonummer ist jetzt optional und die Felder BIC und IBAN sind hinzugekommen und müssen SEPA konform befüllt werden.

Es aber gibt auch eine ganze Reihe neuer Geschäftsvorfälle die mit HKC beginnen. Diese Geschäftsvorfälle haben alle die Versionsnummer 1 und erwarten als Parameter eine SEPA Pain Message, in welcher die Transaktionsdaten enthalten sind. Dazu gehören Überweisungen, terminierte Überweisungen, vorbereitete terminierte Überweisungen, Sammelüberweisungen, terminierte Sammelüberweisungen, in gleicher Form Lastschriften, Lastschriftwiderspruch und Daueraufträge.

Zur Generierung der der SEPA Pain Message (www.iso20022.org) wird ein Generator/Parser zur Verfügung gestellt, der die wesentlichen Inhalt einer SEPA Pain Message parsen und erzeugen kann.

4.9.2 SEPA-fähige Konten

Um zu erkennen, ob ein Konto SEPA-fähig ist, wurde der neue Geschäftsvorfall HKSPA eingeführt.

Mit dem neuen Geschäftsvorfall HKSPA kann zu jedem Konto eine SEPA-Kontoverbindung abgefragt werden. In dieser SEPA Kontoverbindung sind zusätzliche Felder wie BIC/IBAN enthalten.

Wenn in den BPD einer Bank ein HISPAS enthalten ist erneuert die DDBAC automatisch die SEPA Informationen zu jedem Konto. Die geschieht immer dann, wenn ein Kontakt synchronisiert wird oder wenn bei einer Anmeldung neue UPDs zurückgeliefert werden. Die zusätzlichen SEPA Informationen werden in einer separaten Datei im Anwendungsverzeichnis der DDBAC gespeichert. Neben den .bpd, .upd gibt es nun auch .spa Dateien.

In einer .spa Dateien sind die Konten eines Kontakts in XML Form zuzüglich der SEPA Informationen gespeichert:

```
<Accounts>
  <Account>
    <SEPAVerwendung>Y</SEPAVerwendung>
    <AccountNumber>100012345</AccountNumber>
    <CountryCode>280</CountryCode>
    <BankCode>79900010</BankCode>
    <CustomerID>12345</CustomerID>
    <IBAN>DE56799000100100012345</IBAN>
    <BIC>BIC799000</BIC>
  </Account>
  <Account>
    ...
  </Account>
</Accounts>
```

In der DDBAC gibt es bisher das BACAccountData Object mit welchem man auf die UPDs zugreifen konnte. Darin ist aber nur das HIUPD Segment zu jedem Konto (Account) enthalten. Die zusätzlichen Felder BIC und IBAN können nicht über das BACAccountData Object abgefragt werden.

Dazu wurde ein neues BACAccount Object eingeführt. Pro BACContact/BACCustomer können mehrere BACAccount Objecte vorhanden sein, diese können über die neue Accounts-Collection im

BACContact/BACCustomer erhalten werden. Alle Konten zu einem Kontakt können nun beispielsweise so abgefragt werden:

```
foreach (BACAccount aAccount in aBACContact)
{
    String sNum = aAccount.AccountNumber;
    String sName = aAccount.AcctName;
    If (aAccount.IsSEPA())
    {
        String sBIC = aAccount.BIC;
        String sIBAN = aAccount.IBAN;
    }
}
```

Das BACAccount Objekt enthält alle Felder aus dem HIUPD und zusätzlich die Felder für die SEPA Informationen. Siehe auch das „DDBAC Reference Manual.html“. Die BACAccount Collection ist auch für Kontakte ohne SEPA Informationen verfügbar. Die .spa Datei wird dabei automatisch aus den bereits vorhandenen HIUPD Daten erzeugt.

Ein Beispiel für die neuen SEPA Funktionen ist im HBCI.Net Beispiel enthalten. Das Programmierbeispiel HBCI.Net ist im SDK in gezippter Form enthalten. In der gepackten Datei ist eine lauffähige Version und der dazugehörige SourceCode enthalten.

4.9.3 Beispiel: Absenden einer SEPA-Nachricht

Im HBCINet ist ein Beispiel für eine Überweisung mit den neuen SEPA Geschäftsvorfällen enthalten. Die neuen SEPA Geschäftsvorfälle bestehen in der Regel aus Feldern die im HBCI Segment gefüllt werden müssen und einer sepa.pain message im XML Format in welchem die Auftragsdaten enthalten sind.

Das BACSepaMessage Object unterstützt dabei die Generierung der sepa.pain message. Um eine SEPA Überweisung auszuführen muss wie bisher ein BACSegment erstellt werden. Für eine Einzelüberweisung wird der neue HKCCS Geschäftsvorfall verwendet:

```
// Auftragssegment erstellen (Einzelüberweisung)
BACSegment oHKCCS = BAC.CreateTransactionSegment("HKCCS");
```

Um die sepa.pain Message zu erstellen, benötigt man die Klasse BACSepaMessage:

```
// Auftragssegment erstellen (Einzelüberweisung)
BACSegment oHKCCS = BAC.CreateTransactionSegment("HKCCS");
```

Um die sepa.pain Message zu erstellen, benötigt man die Klasse BACSepaMessage:

```
// Sepa pain Message erstellen
BACSepaMessage aSepaMessage = new BACSepaMessage();
```

In diesem BACSepaMessage Object können nun die notwendigen Felder befüllt werden:

```
// Auftrag einstellen
aSepaMessage.Segment = oHKCCS;
```

Hier wird das Auftragssegment der Sepanachricht zugeordnet. Intern kann jetzt bestimmt werden, um welchen Auftrag es sich handelt und welche sepa.pain Nachricht benötigt wird. Dadurch werden intern auch die beiden Felder SEPADescriptor und SEPAPainMessage im BACSegment gefüllt.

```
// Absender einstellen
aSepaMessage.FromAccount = BAC.Account;
```

Jetzt wird der SepaMessage das Auftraggeberkonto übergeben. Dadurch kann die DDBAC im Segment das DEG Auftraggeberkontoverbindung füllen und in der sepa.pain Message die entsprechenden Xml Einträge machen.

```
// Neue Order erstellen (eine für Einzelauftrag, mehrere für
// Sammelüberweisungen)
BACSepaOrder aSepaOrder = aSepaMessage.Orders.Add();
```

Zu der SepaMessage muss nun ein neuer Auftrag (Order) hinzugefügt werden. In Einzelaufträgen ist nur ein Auftrag enthalten bei Sammelaufträgen können beliebig viele Aufträge zu einer SepaMessage hinzugefügt werden.

Jetzt muss ein Empfängerkonto eingerichtet werden. Dazu kann das neue BACAccount Object verwendet werden:

```
// Empfängerkonto eintragen
BACAccount aToAccount = new BACAccount();
aToAccount.CountryCode = "280";
aToAccount.NameEins = txtEmpfaengerName.Text;
aToAccount.AccountNumber = txtEmpfaengerKonto.Text;
aToAccount.BankCode = txtEmpfaengerBLZ.Text;
aToAccount.IBAN = txtIBAN.Text;
aToAccount.BIC = txtBIC.Text;
aSepaOrder.ToAccount = aToAccount;
```

Innerhalb einer sepa.pain Message muss auch der Empfängername angegeben werden, an welchen die Überweisung übermittelt wird.

```
// Empfängername eintragen
aSepaMessage.Recipient = aToAccount;
```

Der Auftrag muss nun mit den üblichen Auftragsdaten befüllt werden:

```
// Betrag
aSepaOrder.Amount = Convert.ToDecimal(txtBetrag.Text);
aSepaOrder.CurrencyCode = "EUR";
// Verwendungszweck
aSepaOrder.Purpose = txtVerwendungszweck1.Text;
```

Für terminierte Überweisungen könnte jetzt auch das Ausführungsdatum übergeben werden:

```
// Optional (für terminierte (Sammel)Überweisungen)
//aSepaMessage.ExcutionDate = "01.01.2007";
```

In sepa.pain XML Dokumenten sind zahlreiche optionale Felder enthalten die nicht direkt durch Properties in der SepaMessage gefüllt werden können. Dennoch können zusätzliche Felder mit SetValue leicht hinzugefügt werden.

```
// Optionale Felder
aSepaOrder.SetValue ("/pfad ", "wert");
```

Mit dem „/pfad“ aus der sepa Spezifikation und dem „wert“ können einzelne XML Werte hinzugefügt werden. Die BACSepaMessage verwendet intern das XML Schema um die XML Nodes an die richtige Position entsprechend des Schemas zu schieben, damit am Ende ein gültiges XML Dokument entsteht.

Dies kann mit dem Aufruf von Verify jederzeit überprüft werden:

```
// Optional: Gegen das Schema verifizieren
String sErrorReason;
aSepaMessage.Verify(out sErrorReason);
if (sErrorReason != null && sErrorReason.Length > 0)
{
    throw new Exception (sErrorReason);
}
```

Wenn in dem Fehlertext sErrorReason ein Text zurückgeliefert wird, dann enthält dieser eine Fehlermeldung des XML Parsers in welchem beschrieben wird, warum das XML nicht dem Schema entspricht. Dieser Fehlertext ist übrigens für Endkunden ungeeignet.

Nachdem alle Felder befüllt wurden kann der Auftrag wie gewohnt ausgeführt werden:

```
// Auftrag ausführen
BACMessage oMessage = BAC.Dialog.ExecuteSegment(oHKCCS) as BACMessage;
BAC.ReportResult (oMessage, oHKCCS);
```

Intern wird dabei das Segment HKCCS aufgebaut und dabei werden die aktuellen Werte der BACSepaMessage in das Feld SEPAPainMessage übertragen und abgesendet. Das XML wird zu diesem Zeitpunkt nicht gegen das Schema geprüft, da es Geschäftsvorfälle wie „Vorbereitete Überweisungen“ gibt, in welchem man unausgefüllte und damit ungültige XML Aufträge versenden darf.

Das BACMessageObject kann auch verwendet werden um empfangene sepa.pain Message zu parsen und die einzelnen Werte aus dem XML zu entnehmen. Ein Beispiel dafür folgt.

Die DDBACXML verarbeitet derzeit noch keine SEPA-Aufträge.

5 Erweiterung/Ausbau der DDBAC

5.1 Das neue HBCI-Syntaxobjektmodell

Seit der DDBAC-Version 2.0.9 gibt es ein Objektmodell für die Segmentsyntax. Dieses Objektmodell enthält Informationen über die Segmentsyntax. Es bietet Methoden und Eigenschaften um nach einem Segment oder einem Element zu suchen und um bestehende Syntaxbeschreibungen zu ändern bzw. neue Syntaxbeschreibungen hinzuzufügen. Detaillierte Beschreibung der Methoden und Eigenschaften befinden sich im [DDBAC Reference Manual.html](#). Anbei eine Übersicht des Objektmodells.

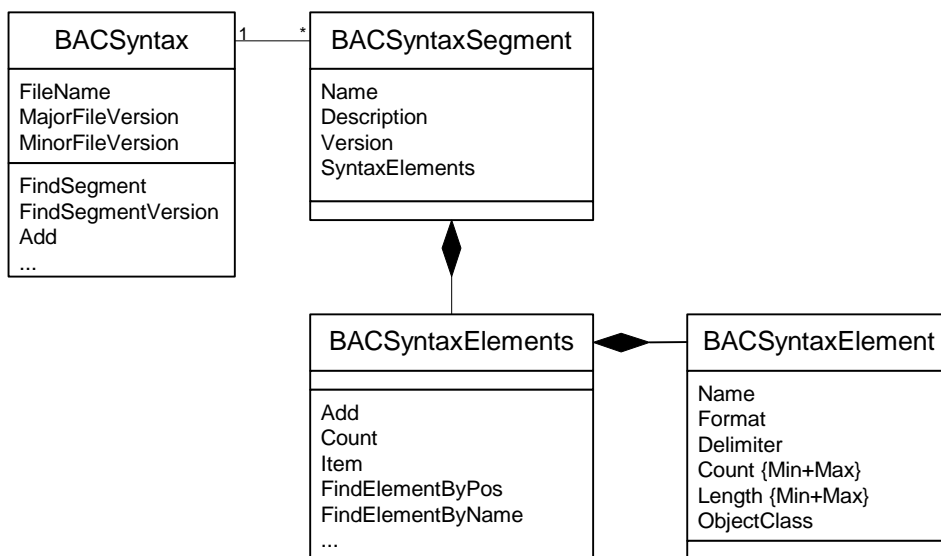


Abb. 4 - Klassendiagramm des Syntaxobjektmodells in der UML-Darstellung

Das BACSyntax-Objekt ist das Stammobjekt der Syntaxrepräsentation. Es bietet eine Methode, um neue Segmentsyntaxen (Add) hinzuzufügen und Methoden, um eine vorhandene Segmentsyntax zu finden.

Das BACSyntaxSegment-Objekt ist ein Container mit Eigenschaften, die das Segment beschreiben. Die Eigenschaft "SyntaxElements" enthält ein BACSyntaxElements-Objekt.

Das BACSyntaxElements-Objekt ist eine Collection der BACSyntaxElement-Objekte. Das BACSyntaxElements-Objekt beschreibt die Syntax des ganzen Segments. Dieses Objekt ermöglicht den Zugang zu den individuellen BACSyntaxElement-Objekten.

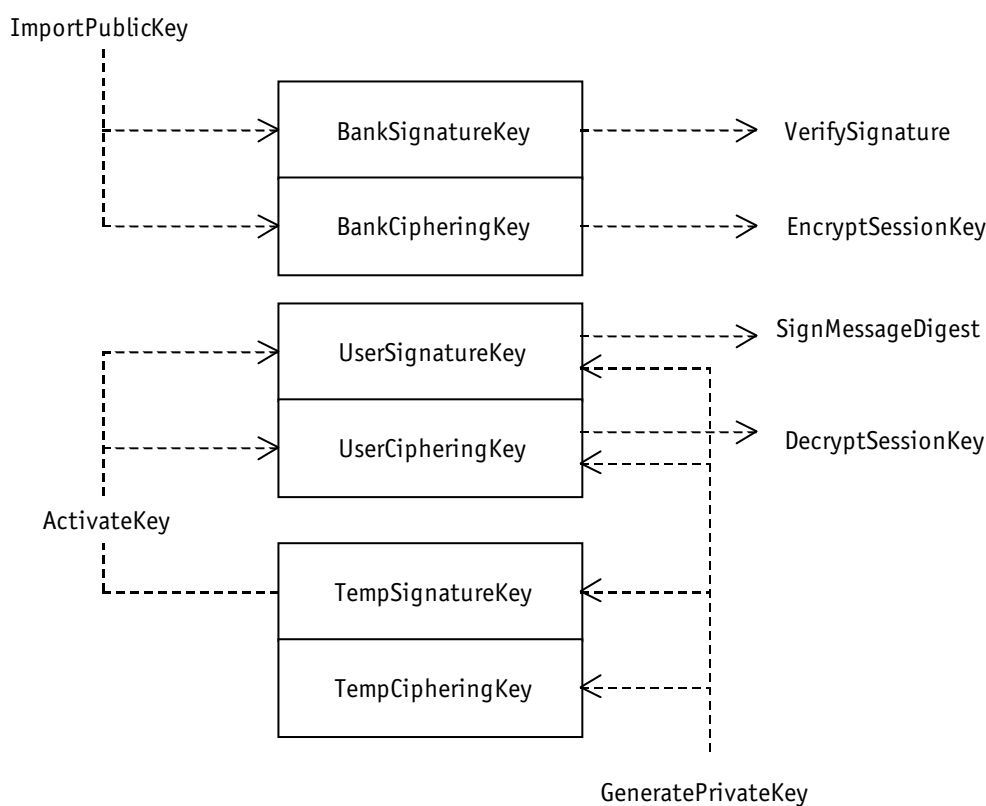
Das BACSyntaxElement beschreibt die Syntax eines Elementes. Ein Element kann entweder ein einfaches Element, wie z.B. das numerische Element "Segmentnummer" im Segmenkopf, sein oder eine Beschreibung einer Elementengruppe, wie z.B. "Segmentkennung" im Segmentkopf.

5.2 Erstellung eines neuen Security-Objektes

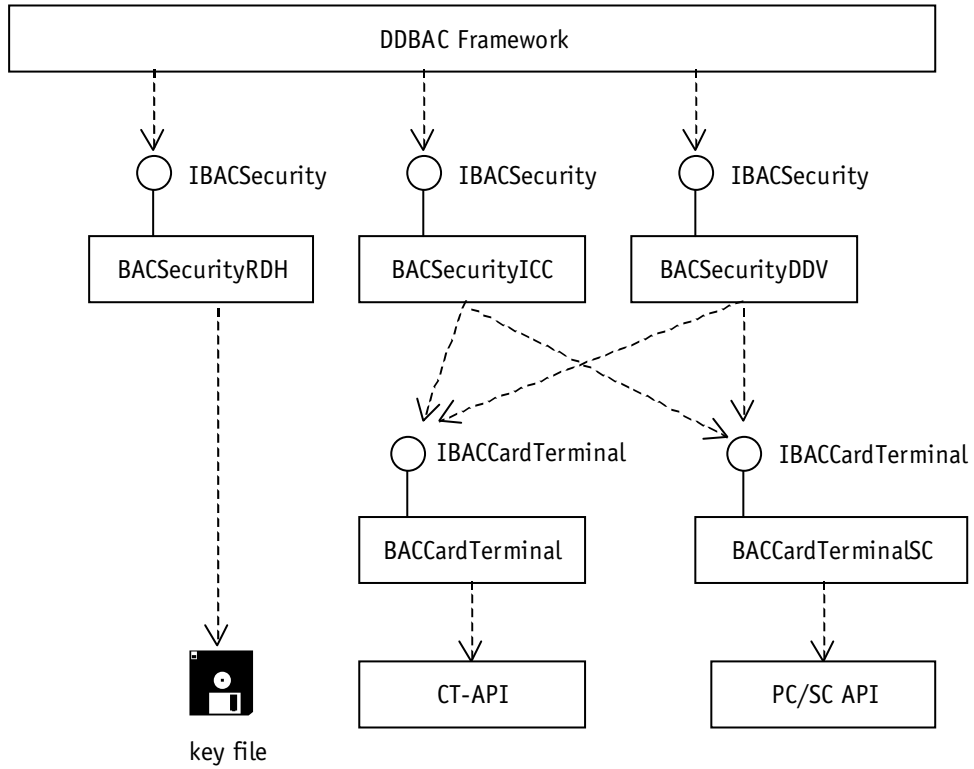
Die DDBAC definiert ein allgemeines Security-Modell. Mit diesem Modell ist es möglich neue Sicherheitsmedien zum DDBAC-Framework hinzuzufügen. Ein Security-Objekt ist ein Modell eines HBCI-Sicherheitsmediums, wie z.B. einer ChipCard oder eines Schlüsseldiskette.

Alle DDBAC Security-Objekte müssen eine Implementierung des polymorphen Interfaces IBACSecurity besitzen. Alle vom DDBAC-Framework geforderten Sicherheitsarbeitsabläufe werden durch die Methoden dieses Interfaces ausgeführt.

Das Modell des Security-Objektes erlaubt die Abspeicherung von bis zu sechs Schlüsseln, die durch ihre Key IDs identifiziert werden können, voraus. Symmetrische Sicherheitsmethoden speichern nur zwei Schlüssel ab. Die folgende Abbildung zeigt, wie diese Schlüssel von den IBACSecurity-Methoden verwaltet werden. Die Methode ExportPublicKey kann verwendet werden, um die jeweiligen öffentlichen Teile aller sechs Schlüssel zu exportieren.



Jeder Schlüssel ist einem Schlüssel-Namen gemäß HBCI zugewiesen. Der key-Name wird dem Moment zugeteilt, zu dem der Schlüssel der Security-Methode bekannt wird, d.h., er wird durch einen Parameter der Methode ImportPublicKey zugeteilt oder automatisch abgeleitet durch den GeneratePrivateKey. Die Implementierung muss imstande sein, den key-Namen aller sechs Schlüssel zu unterstützen.



6 Weitere Informationen

6.1 Beispielsapplikationen

Das DDBAC-SDK enthält vier Beispielapplikationen HBCIPAD, HBCINET, HBCIPING und HBCISyntax. Alle Beispielapplikationen sind auch als Sourcecode vorhanden.

Die Beispielapplikation HBCIPAD ist eine einfache Homebanking-Applikation, in Visual Basic geschrieben. Es gibt auch das HBCINET, das ähnlich wie HBCIPAD arbeitet, aber in C# geschrieben ist.

Das Programm „HBCIPING“ ist eine minimale Anwendung mit der die Verfügbarkeit eines HBCI Banksystems anonym getestet werden kann.

Hinweis

Die Beispielprogramme für das .Net-Framework finden Sie im DDBACSDK-Verzeichnis (in der Regel c:\Programme\DataDesign\DDBACSDK) in gezippten Paketen.

Die Beispielprogramme HBCIPad, HBCIPing und HBCISyntax können Sie über das Startmenü aufrufen. Den SourceCode der Beispielprogramme finden Sie in den jeweiligen Unterverzeichnissen unterhalb des DDBACSDK-Verzeichnis (in der Regel c:\Programme\DataDesign\DDBACSDK).

6.2 Segmentbeschreibungen

Eine tabellarische Beschreibung aller HBCI Segmente die derzeit von den DDBAC unterstützt werden, liegt in der Online Datei „DDBAC Segmente.html“ vor.

6.3 DDBAC Referenzhandbücher

Die vollständige (derzeit englischsprachige) Referenz zu den Schnittstellen der DataDesign HBCI Banking Application Components liegt in der Online Datei „DDBAC Reference Manual.html“ vor.

Die vollständige Referenz zu den Schnittstellen der SWIFT Format Komponente liegt in der Online Datei „DDBAC SWIFT Referenzhandbuch.html“ vor.

Die vollständige Referenz zu den Schnittstellen der DTA Format Komponente liegt in der Online Datei „DDBAC DTA Referenzhandbuch.html“ vor.

Das Referenzhandbuch zum Format der RDH-Sicherheitsdatei ist auf Anfrage verfügbar.

Die Spezifikation von FOAM sowie der DDBAC-XML-Schnittstellen sind ebenfalls auf Anfrage verfügbar.