

Net.Data



Verwaltung und Programmierung für OS/400

Net.Data



Verwaltung und Programmierung für OS/400

Version 2 Release 2

Anmerkung

Vor Verwendung dieser Informationen und des darin beschriebenen Produkts sollten Sie die allgemeinen Informationen in Anhang C, „Bemerkungen“ auf Seite 145 lesen.

Diese Veröffentlichung ist eine Übersetzung des Handbuchs
Net.Data Administration and Programming Guide for OS/400,

herausgegeben von International Business Machines Corporation, USA
© Copyright International Business Machines Corporation 1999

© Copyright IBM Deutschland Informationssysteme GmbH 1999

Informationen, die nur für bestimmte Länder Gültigkeit haben und für Deutschland, Österreich und die Schweiz nicht zutreffen, wurden in dieser Veröffentlichung im Originaltext übernommen.

Möglicherweise sind nicht alle in dieser Übersetzung aufgeführten Produkte in Deutschland angekündigt und verfügbar; vor Entscheidungen empfiehlt sich der Kontakt mit der zuständigen IBM Geschäftsstelle.

Änderung des Textes bleibt vorbehalten.

Herausgegeben von:
SW NLS Center
Kst. 2877
Juni 1999

Inhaltsverzeichnis

Vorwort	vii
Informationen zu Net.Data	vii
Neue Funktionen in diesem Release	viii
Informationen zu diesem Handbuch	ix
Zielgruppe	ix
Informationen zu Beispielen in diesem Handbuch	ix
 Einführung	1
Net.Data - Produktbeschreibung	1
Gründe zur Verwendung von Net.Data	2
 Konfigurieren von Net.Data	5
Kopieren des Net.Data-Programmdateiobjekts in Ihre CGI-BIN-Bibliothek	5
Informationen zur Net.Data-Initialisierungsdatei	6
Anpassen der Net.Data-Initialisierungsdatei	7
Erstellen einer Initialisierungsdatei	8
Konfigurationsvariablenanweisungen	8
Pfadkonfigurationsanweisungen	15
Umgebungs-konfigurationsanweisungen	20
Einrichten der Sprachumgebungen	22
Definieren der Java-Anwendungssprachumgebung	22
Definieren der SQL-Sprachumgebung	23
Konfigurieren des Web-Servers	23
Erteilen von Zugriffsrechten für Objekte, auf die Net.Data zugreift	25
 Sichern der Datenbestände	27
Verwenden von Firewalls	27
Verschlüsseln Ihrer Daten im Netzwerk	29
Verwenden der Authentifizierung	30
Verwenden der Berechtigung	31
Verwenden von Net.Data-Mechanismen	31
Net.Data-Konfigurationsvariablen	31
Makro-Entwicklungsverfahren	32
 Aufrufen von Net.Data	37
Aufrufen von Net.Data mit einem Makro (Makroanforderung)	37
HTML-Programmverbindungen (Links)	39
HTML-Formulare	39
Aufrufen eines permanenten Makros	40
Syntax permanenter Makros	41
Beispiele	42
 Entwickeln von Net.Data-Makros	43
Aufbau eines Net.Data-Makros	44
Der DEFINE-Block	46
Der FUNCTION-Block	47
HTML-Blöcke	48
Net.Data-Makrovariablen	51
Geltungsbereich von Kennungen	51
Definieren von Variablen	53

Verweisen auf Variablen	55
Variablenarten	57
Net.Data-Funktionen	66
Definieren von Funktionen	66
Aufrufen von Funktionen	72
Aufrufen von integrierten Net.Data-Funktionen	72
Generieren von Web-Seiten in einem Makro	77
HTML-Blöcke	77
REPORT-Blöcke	79
Bedingte Logik und Schleifen in einem Makro	86
Bedingte Logik: IF-Blöcke	86
Schleifenkonstrukte: WHILE-Blöcke	90
Verwenden der Sprachumgebungen	91
Übersicht über die von Net.Data bereitgestellten Sprachumgebungen	92
Aufrufen einer Sprachumgebung	92
Behandeln von Fehlerbedingungen	93
Sicherheit	93
Direktaufrufsprachumgebung	93
Aufrufen von Programmen	93
Übergeben von Parametern an Programme	94
Zurückgeben von Werten aus Programmen	97
Beispiel für die Direktaufrufsprachumgebung	97
Java-Anwendungssprachumgebung	98
Aufrufen von Java-Programmen	98
Übergeben von Parametern an Java-Programme	99
Beispiel für die Java-Anwendungssprachumgebung	99
REXX-Sprachumgebung	99
Ausführen von REXX-Programmen	99
Übergeben von Parametern an REXX-Programme	100
Beispiel für die REXX-Sprachumgebung	102
SQL-Sprachumgebung	102
Ausführen von SQL-Anweisungen	102
Überlegungen zu Datentypen	105
Verwalten von Transaktionen in einer Net.Data-Anwendung	110
Verwalten von Mehrfachdatenbankverbindungen	111
Gespeicherte Prozeduren	112
Beispiel für die SQL-Sprachumgebung	118
SYSTEM-Sprachumgebung	120
Absetzen von Befehlen und Aufrufen von Programmen	120
Übergeben von Parametern an Programme	120
Beispiel für die SYSTEM-Sprachumgebung	122
Transaktionsverwaltung mit permanenten Makros	123
Informationen zu permanenten Makros	123
Definieren einer Transaktion	124
Starten einer Transaktion	125
Angaben der Makro-HTML-Blöcke in einer Transaktion	126
Beenden einer Transaktion	130
Definieren des Gültigkeitsbereichs einer Variablen in einer Transaktion	130
Angaben von COMMIT- und ROLLBACK-Operationen in einer Transaktion	131
Beispiel für ein permanentes Makro	132

	Optimieren der Leistung	135
	Net.Data-Caching von Makros	135
	Optimieren der Sprachumgebungen	135
	REXX-Sprachumgebung	135
	SQL-Sprachumgebung	136
	SYSTEM-Sprachumgebung	137
	 Anhang A. Literaturübersicht	139
	Net.Data Technical Library	139
	Referenzliteratur	139
	 Anhang B. Net.Data-Beispielmakro	141
	 Anhang C. Bemerkungen	145
	Änderungen in der IBM Terminologie	147
	Marken	148
	 Glossar	149
	 Index	151

Vorwort

Vielen Dank, daß Sie sich für Net.Data, das IBM Entwicklungs-Tool zum Erstellen dynamischer Web-Pages, entschieden haben! Mit Hilfe von Net.Data können Sie schnell Web-Seiten mit dynamischem Inhalt entwickeln, indem Sie Daten aus einer Vielzahl von Datenquellen integrieren und die Leistungsstärke der Ihnen bereits bekannten Programmiersprachen ausschöpfen.

Informationen zu Net.Data

Mit Net.Data von IBM können Sie unter Verwendung von Daten aus relationalen und nichtrelationalen Datenbankverwaltungssystemen (DBMS - Database Management Systems), einschließlich DB2-Datenbanken, auf die über DRDA zugegriffen werden kann, sowie unter Verwendung von Anwendungen, die in Programmiersprachen wie Java, JavaScript, Perl, C, C++ und REXX geschrieben wurden, dynamische Web-Pages erstellen.

Net.Data ist ein Makroumwandler, der als Middleware auf einer Web-Server-Maschine ausgeführt wird. Sie können Net.Data-Anwendungsprogramme (sogenannte *Makros*) schreiben, die von Net.Data interpretiert und für die Erstellung dynamischer Web-Pages mit angepaßtem Inhalt verwendet werden. Grundlage hierfür sind die Eingabe vom Benutzer, der aktuelle Status Ihrer Datenbanken, andere Datenquellen, vorhandene Geschäftslogik und andere Faktoren, die Sie in den Makroentwurf aufnehmen.

Eine Anforderung in Form einer URL-Adresse (URL - Uniform Resource Locator) wird von einem Browser, wie Netscape Navigator oder Internet Explorer, an einen Web-Server gesendet, der die Anforderung zur Ausführung an Net.Data weiterleitet. Net.Data lokalisiert das Makro, führt es aus und erstellt eine Web-Seite, die basierend auf den von Ihnen definierten Funktionen angepaßt wird. Diese Funktionen können folgende Aktionen ausführen:

- Einbinden von Geschäftslogik in Anwendungen, die u. a. in den Programmiersprachen C, C++, RPG, COBOL, Java oder REXX geschrieben sind
- Zugreifen auf Datenbanken wie DB2
- Zugreifen auf andere Datenquellen wie unstrukturierte Textdateien

Net.Data gibt diese Web-Seite an den Web-Server weiter, der die Seite seinerseits über das Netzwerk zur Anzeige im Browser weiterleitet.

Net.Data kann in Server-Umgebungen verwendet werden, die zur Verwendung von Schnittstellen wie HTTP (HyperText Transfer Protocol) und CGI (Common Gateway Interface) konfiguriert sind. HTTP ist eine dem Industriestandard entsprechende Schnittstelle für die Interaktion zwischen einem Browser und einem Web-Server, und CGI ist eine dem Industriestandard entsprechende Schnittstelle für den Web-Server-Aufruf von Gateway-Anwendungen wie Net.Data. Diese Schnittstellen ermöglichen Ihnen die Auswahl Ihres bevorzugten Browsers bzw. Web-Servers zur Verwendung mit Net.Data. Net.Data unterstützt außerdem eine Vielzahl von Web-Server-APIs (Application Programming Interfaces - Anwendungsprogrammierschnittstellen) für verbesserte Leistung. Die Net.Data-Produktfamilie bietet ein ähnliches Leistungsspektrum unter OS/400, OS/390, Windows NT, AIX, OS/2, HP-UX, Sun Solaris, Linux und SCO (Santa Cruz Operating System).

Neue Funktionen in diesem Release

Net.Data für OS/400 weist in diesem Release die folgenden neuen Funktionen auf:

- Verbesserte Leistung durch die Fähigkeit, Makros und Kopfdateien zwischenspeichern
- Die Sprachumgebungen (Language Environment) wurden unter anderem wie folgt erweitert:
 - Zwei neue Sprachumgebungen:
 - Java-Anwendung
 - Direktaufruf
 - Unterstützung für große Objekte (LOBs) in der SQL-Sprachumgebung
- Die Makrosprache wurde unter anderem wie folgt erweitert:
 - Fähigkeit, E-Mail-Nachrichten zu generieren und mit der integrierten Funktion DTW_SENDMAIL vom Makro aus zu senden
 - Fähigkeit, HTTP-Cookies mit den integrierten Funktionen DTW_GETCOOKIE und DTW_SETCOOKIE abzurufen und festzulegen
 - Fähigkeit, Zeichenfolgen mit der Funktion DTW_REPLACE zu ersetzen
 - Unterstützung für Millisekunden in der Funktion DTW_TIME
 - Unterstützung für das dynamische Erstellen von Variablenverweisen
 - Fähigkeit, das Attribut VALUE im Element OPTION von DTW_SELECT() festzulegen
 - Unterstützung für das Schlüsselwort RETURNS im Sprachkonstrukt MACRO_FUNCTION
 - Unterstützung für das Hash-Zeichen (#) in Variablennamen
- Die Konfiguration wurde unter anderem wie folgt erweitert:
 - Bei von Net.Data bereitgestellten Sprachumgebungen sind in der Net.Data-Initialisierungsdatei keine ENVIRONMENT-Anweisungen mehr erforderlich.
 - Fähigkeit, die Variable SHOWSQL mit der Konfigurationsvariablen DTW_SHOWSQL zu inaktivieren (standardmäßig inaktiviert)

Informationen zu diesem Handbuch

In diesem Handbuch werden Verwaltungs- und Programmierungskonzepte für Net.Data sowie das Konfigurieren von Net.Data und seiner Komponenten, das Planen der zu verwendenden Sicherheitsmaßnahmen sowie Maßnahmen zur Steigerung der Systemleistung erläutert.

Auf Ihrer Kenntnis von Programmiersprachen und Datenbanken aufbauend lernen Sie die Verwendung der Net.Data-Makrosprache zum Entwickeln von Makros. Sie lernen die Verwendung der von Net.Data bereitgestellten Sprachumgebungen, die auf DB2-Datenbanken und Sie werden in die Verwendung von RPG, COBOL und anderen Programmiersprachen zum Zugriff auf Ihre Daten eingeführt.

In diesem Handbuch wird möglicherweise auf angekündigte, jedoch noch nicht allgemein verfügbare Produkte und Funktionen verwiesen.

Weitere Informationen wie Net.Data-Beispielmakros, Demos und die neueste Version dieses Handbuchs können über folgende World Wide Web-Sites abgerufen werden:

<http://www.software.ibm.com/data/net.data>

<http://www.as400.ibm.com/netdata>

Zielgruppe

Dieses Handbuch richtet sich an Planer und Programmierer von Net.Data-Anwendungen. Als Grundlage zum Verständnis der in diesem Handbuch erläuterten Konzepte müssen Sie damit vertraut sein, wie ein Web-Server funktioniert, einfache SQL-Anweisungen verstehen und HTML-Befehle, einschließlich HTML-Formularbefehle, kennen.

Die Makrosprache, Variablen und integrierten Funktionen in Net.Data sowie die Unterschiede zwischen den einzelnen Betriebssystemen werden im Handbuch *Net.Data Reference* beschrieben.

Informationen zu Beispielen in diesem Handbuch

Die in diesem Handbuch verwendeten Beispiele sind möglichst einfach gehalten, um bestimmte Konzepte darzustellen. Sie zeigen nicht jede Möglichkeit für den Einsatz von Net.Data-Konstrukten. Einige Beispiele zeigen nur Ausschnitte, bei denen für eine Ausführung zusätzlicher Code erforderlich ist.

Einführung

Die meisten Web-Seiten im Internet sind statische Web-Seiten, d. h., sie ändern sich nur, wenn sie von Ihnen editiert werden. Sollen dynamische Daten und Anwendungen (z. B. aktuelle Verkaufsstatistiken) ins Web gestellt werden, schreiben Web-Site-Entwickler in der Regel Programme, die auf dem Web-Server als Middleware ausgeführt werden, um dynamisch Web-Seiten zu erstellen. Das Schreiben solcher Programme ist allerdings nicht ganz einfach.

Net.Data vereinfacht das Schreiben interaktiver Web-Anwendungen durch *Makros*.

In diesem Kapitel wird Net.Data beschrieben und werden die Gründe zur Verwendung für Ihre Web-Anwendung genannt.

- „Net.Data - Produktbeschreibung“
- „Gründe zur Verwendung von Net.Data“ auf Seite 2

Net.Data - Produktbeschreibung

Mit Hilfe von Net.Data-Makros können Sie Programmierungslogik ausführen, auf Variablen zugreifen und sie bearbeiten, Funktionen aufrufen und Tools zum Generieren von Berichten verwenden. Ein Makro ist eine Textdatei mit Net.Data-Makrosprachkonstrukten, HTML-Befehlen, JavaScript und Anweisungen einer Sprachumgebung wie SQL. Net.Data verarbeitet das Makro, um eine Ausgabe, die von einem Web-Browser angezeigt werden kann, zu erzeugen.

Makros kombinieren die Einfachheit von HTML mit der dynamischen Funktionalität von Web-Server-Programmen, wodurch das Hinzufügen von dynamischen Daten zu statischen Web-Seiten erheblich vereinfacht wird. Die dynamischen Daten können aus lokalen bzw. fernen Datenbanken und aus unstrukturierten Textdateien extrahiert oder durch Anwendungen und Systemservices generiert werden.

Abb. 1 auf Seite 2 illustriert die Beziehung zwischen Net.Data für OS/400, dem Web-Server und unterstützten Daten sowie Programmiersprachumgebungen.

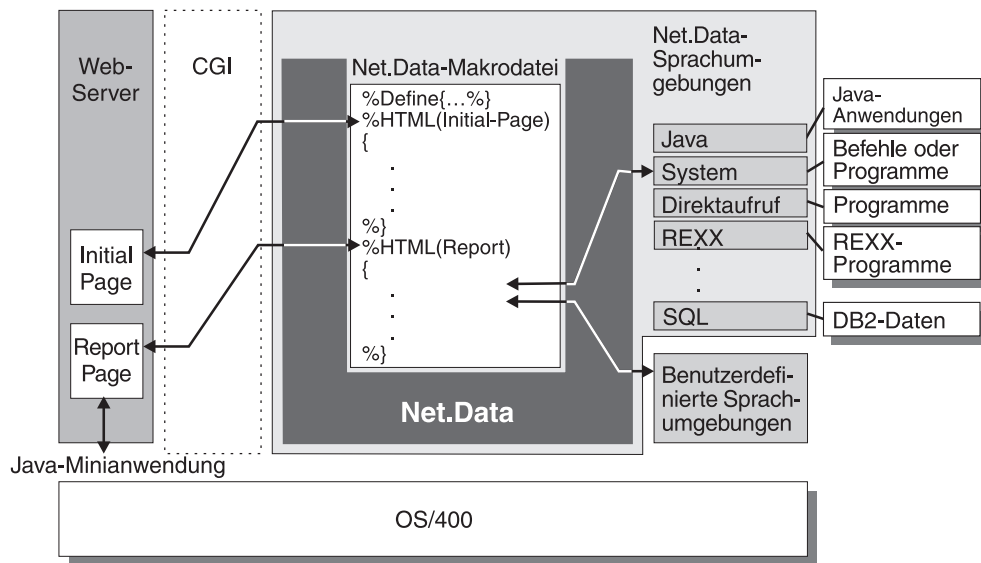


Abbildung 1. Die Beziehung zwischen Net.Data für OS/400, dem Web-Server und unterstützten Daten sowie Programmquellen

Der Web-Server ruft Net.Data als eine CGI-Anwendung auf, wenn eine URL-Adresse empfangen wird, die Net.Data-Services anfordert. Die URL-Adresse enthält Net.Data-spezifische Informationen, und zwar das zu verarbeitende Makro. Wenn Net.Data die Verarbeitung der Anforderung beendet hat, wird die entstandene Web-Seite an den Web-Server gesendet. Der Server übergibt diese an den Web-Client, auf dem sie mit dem Browser angezeigt wird.

Gründe zur Verwendung von Net.Data

Net.Data ist eine gute Wahl für das Erstellen dynamischer Web-Pages, weil die Verwendung der Makrosprache einfacher ist als das Schreiben eigener Web-Server-Anwendungen und weil Net.Data den Einsatz von Sprachen ermöglicht, die Ihnen bereits bekannt sind, wie HTML, SQL, REXX und JavaScript. Net.Data stellt zudem Sprachumgebungen (Language Environments) bereit, die auf DB2-Datenbanken zugreifen, oder REXX, Perl und andere Sprachen für Ihre Anwendungen verwenden. Außerdem werden Änderungen an einem Makro sofort in einem Browser angezeigt.

Net.Data erweitert die Datenverwaltungsfunktionen, die bereits für Ihr Betriebssystem entwickelt wurden, indem sowohl Daten als auch die zugehörige Geschäftslogik für das Web aktiviert werden. Im einzelnen gilt für Net.Data folgendes:

- Es wird eine einfache, jedoch leistungsfähige Makrosprache bereitgestellt, die die rasche Entwicklung von Internet- und Intranet-Anwendungen ermöglicht. Die Net.Data-Web-Anwendungsumgebung bietet die folgenden Funktionen:
- In Ihren Web-Anwendungen kann die Datengenerierungslogik von der Darstellungslogik getrennt werden. Bei Net.Data gibt es keine Einschränkungen hinsichtlich der Darstellungsmethode für die Daten (z. B. HTML oder JavaScript). Durch diese Trennung können die Benutzer die Darstellung der Daten leicht entsprechend den neuesten Darstellungstechniken ändern.

- Vorhandene Erfahrung und Geschäftslogik können zum Generieren von Web-Pages verwendet werden, indem Programme, die in C, C++, RPG, COBOL, REXX, Java oder anderen Sprachen geschrieben wurden, eingebunden werden können.
- Komplexe Internet-Anwendungen können mit Hilfe einer einfachen Makrosprache schnell entwickelt werden.
- Es wird ein leistungsfähiger Zugriff auf Daten ermöglicht, die in DB2 und in jeder beliebigen fernen DRDA-fähigen Datenbank gespeichert sind.
- Makros können mühelos zwischen allen von der Net.Data-Produktfamilie unterstützten Betriebssystemen migriert werden.

Interpreter-Makrosprache

Die Net.Data-Makrosprache ist eine Interpreter-Sprache. Wenn Net.Data zur Verarbeitung eines Makros aufgerufen wird, interpretiert Net.Data direkt jede Sprachanweisung und zwar sequentiell oben in der Datei beginnend. Bei dieser Vorgehensweise können Sie am Makro vorgenommene Änderungen bei der nächsten Angabe der URL-Adresse, die das Makro ausführt, sofort sehen. Eine Neukompilierung ist nicht erforderlich.

Freies Format

Die Net.Data-Makrosprache weist nur einige wenige Regeln zum Programmformat auf. Diese Unkompliziertheit ermöglicht Programmierern ein hohes Maß an Freiheit und Flexibilität. Eine einzelne Anweisung kann sich über mehrere Zeilen erstrecken, oder mehrere Anweisungen können auf einer einzelnen Zeile eingegeben werden. Die Anweisungen können in einer beliebigen Spalte beginnen. Hierbei können Leerzeichen und sogar ganze Zeilen übersprungen werden. Kommentare können an einer beliebigen Stelle eingefügt werden.

Variablen ohne Typ

Net.Data interpretiert alle Daten als Zeichenfolgen. Net.Data führt mit integrierten Funktionen arithmetische Operationen für eine Zeichenfolge aus, die eine gültige Zahl darstellt, einschließlich jener in Exponentialschreibweise. Die Variablen der Makrosprache werden in „Net.Data-Makrovariablen“ auf Seite 51 näher beschrieben.

Integrierte Funktionen

Net.Data verfügt über integrierte Funktionen, die verschiedene Verarbeitungsprozesse, Such- und Vergleichsoperationen sowohl für Text als auch für Zahlen ausführen. Andere integrierte Funktionen bieten Unterstützung bei der Formatierung und bei arithmetischen Berechnungen.

Fehlerbehandlung

Wenn Net.Data einen Fehler feststellt, werden entsprechende Nachrichten mit Erklärungen an den Client zurückgegeben. Sie können die Fehlernachrichten anpassen, bevor sie über einen Browser an einen Benutzer zurückgegeben werden. Weitere Informationen finden Sie im Handbuch *Net.Data Reference*.

Konfigurieren von Net.Data

Net.Data für OS/400 wird als Standardkomponente in folgenden Paketen geliefert:

- IBM TCP/IP Connectivity Utilities/400 V3R2, V3R7, V4R1 und V4R2
- IBM HTTP Server for AS/400 V4R3 und nachfolgende Releases

Es müssen keine weiteren Komponenten gekauft werden, und Sie müssen keine Net.Data-Software herunterladen und installieren.

Die AS/400-TCP/IP- und HTTP Server-Software, die Sie benötigen, ist im Lieferumfang von OS/400 enthalten, wird jedoch nur wahlfrei installiert. Die folgende wahlfrei installierbare Software muß für die folgenden Versionen des Betriebssystems OS/400 auf Ihrem System installiert werden:

- Für das Betriebssystem IBM OS/400 Version 3 Release 2, Version 3 Release 7 und nachfolgende Versionen und Releases (57xx-SS1):
 - IBM TCP/IP Connectivity Utilities/400 (57xx-TC1)
- Für das Betriebssystem IBM OS/400 Version 4 Release 3 und nachfolgende Versionen und Releases (57xx-SS1):
 - IBM HTTP Server for AS/400 (57xx-DG1)

Führen Sie nach der Installation von Net.Data die in den folgenden Abschnitten beschriebenen Schritte aus, um Net.Data für OS/400 zu konfigurieren:

- „Kopieren des Net.Data-Programmdateiobjekts in Ihre CGI-BIN-Bibliothek“
- „Erstellen einer Initialisierungsdatei“ auf Seite 8
- „Anpassen der Net.Data-Initialisierungsdatei“ auf Seite 7
- „Einrichten der Sprachumgebungen“ auf Seite 22
- „Konfigurieren des Web-Servers“ auf Seite 23
- „Erteilen von Zugriffsrechten für Objekte, auf die Net.Data zugreift“ auf Seite 25

Kopieren des Net.Data-Programmdateiobjekts in Ihre CGI-BIN-Bibliothek

Bevor Sie Net.Data verwenden, müssen Sie das Net.Data-Programmdateiobjekt in die CGI-BIN-Bibliothek kopieren und Zugriffsrechte für das Objekt vergeben.

Gehen Sie wie folgt vor, um das Net.Data-Programmdateiobjekt zu kopieren:

1. Verwenden Sie den Befehl CRTDUPOBJ (Doppeltes Objekt erstellen), und kopieren Sie das Net.Data-Programmdateiobjekt, DB2WWW, aus der Bibliothek QTCP in eine CGI-BIN-Bibliothek.

Benutzer von OS/400 V4R3: Verwenden Sie das Programmdateiobjekt in der Bibliothek QHTTPSVR; das Programmdateiobjekt in der Bibliothek QTCP leitet Net.Data-Anforderungen an die Bibliothek QHTTPSVR weiter.

2. Ändern Sie das Programmdateiobjekt DB2WWW im CGI-BIN-Verzeichnis so, daß das Benutzerprofil, unter dem CGI-Programme ausgeführt werden, Zugriff auf das Programmdateiobjekt hat.

In der Standardeinstellung ist die Berechtigung des Programmdateiobjekts DB2WWW für *PUBLIC-Benutzer auf *EXCLUDE gesetzt. Ändern Sie die Berechtigung des Programmdateiobjekts für *PUBLIC-Benutzer in *USE, oder geben Sie dem Benutzerprofil explizit Zugriff auf das Programmdateiobjekt DB2WWW, um den Zugriff auf das Programmdateiobjekt zu ermöglichen.

Sie können das Net.Data-Programmdateiobjekt für verschiedene Anwendungen in mehrere Bibliotheken kopieren. So können Sie mehrere Versionen der Net.Data-Initialisierungsdatei oder mehrere Zugriffsschutzvarianten verwenden.

Weitere Informationen zur Net.Data-Initialisierungsdatei finden Sie in „Anpassen der Net.Data-Initialisierungsdatei“ auf Seite 7. Informationen zur Authentifizierung finden Sie in „Verwenden der Authentifizierung“ auf Seite 30.

Gehen Sie wie folgt vor, um das Net.Data-Programmdateiobjekt in mehrere Bibliotheken zu kopieren:

1. Kopieren Sie das Net.Data-Programmdateiobjekt, DB2WWW, in eine Bibliothek. Führen Sie dazu die oben genannten Schritte aus.
2. Ordnen Sie dem Net.Data-Programmdateiobjekt ein CL-Programm in jeder Bibliothek zu.
 - a. Erstellen Sie ein CL-Programm, das das Net.Data-Programmdateiobjekt aufruft, das sich in der Bibliothek befindet, die in Schritt 1 angegeben wurde.
 - b. Kopieren Sie das CL-Programm in jede Bibliothek.

Das von Ihnen erstellte CL-Programm wird so zum Net.Data-Programmdateiobjekt. Wenn Sie dem Programmdateiobjekt kein CL-Programm zuordnen und das Net.Data-Programmdateiobjekt DB2WWW in verschiedene Bibliotheken kopieren, erhalten Sie den SQL-Fehlercode -901, wenn Sie die SQL-Sprachumgebung verwenden.

In den folgenden Abschnitten sollte das von Ihnen erstellte CL-Programm mit dem Net.Data-Programmdateiobjekt gleichgesetzt werden, wenn Sie das CL-Programm so erstellt haben, daß Net.Data aufgerufen wird.

Informationen zur Net.Data-Initialisierungsdatei

Net.Data verwendet seine Initialisierungsdatei zum Festlegen der Einstellungen verschiedener Konfigurationsvariablen und zum Konfigurieren der Sprachumgebungen und Suchpfade. Die Einstellungen von Konfigurationsvariablen steuern verschiedene Aspekte der Net.Data-Operation, unter anderem:

- Die Angabe eines SMTP-Servers und eines Zeichensatzes zum Senden von E-Mail
- Die Aktivierung der SQL-Sprachumgebungsvariablen SHOWSQL

Die Sprachumgebungsanweisungen definieren die Net.Data-Sprachumgebungen, die verfügbar sind, und geben spezielle Eingabe- und Ausgabeparameterwerte an, die mit den Sprachumgebungen ausgetauscht werden. Die Sprachumgebungen ermöglichen es Net.Data, auf verschiedene Datenquellen, z. B. DB2-Datenbanken und Systemservices, zuzugreifen. Die Pfadanweisungen geben die Verzeichnispfade zu Dateien an, die Net.Data verwendet, z. B. Makros und Programme .

Die Erstellung der Net.Data-Initialisierungsdatei ist bei OS/400 wahlfrei. Durch Verwendung einer Initialisierungsdatei können Sie in Ihren Net.Data-Makrodateien kürzere URL-Adressen und kürzere Verweise auf Programme und Kopfdateien verwenden. Eine Initialisierungsdatei ist jedoch erforderlich, wenn Sie eine eigene Sprachumgebung erstellen möchten.

Wenn Sie keine Initialisierungsdatei erstellen, wird Net.Data so ausgeführt, als hätten Sie eine Initialisierungsdatei mit nur den unterstützten Sprachumgebungsanweisungen konfiguriert (Informationen zu den unterstützten Sprachumgebungen finden Sie in „Verwenden der Sprachumgebungen“ auf Seite 91). In diesem Fall müssen alle Verweise auf Makros, Kopfdateien und Programme im Makro mit dem vollständig qualifizierten Pfad angegeben werden.

Anpassen der Net.Data-Initialisierungsdatei

Die in der Initialisierungsdatei enthaltenen Informationen werden mit drei Arten von Konfigurationsanweisungen angegeben, die in den folgenden Abschnitten beschrieben werden:

- „Konfigurationsvariablenanweisungen“ auf Seite 8
- „Pfadkonfigurationsanweisungen“ auf Seite 15
- „Umgebungskonfigurationsanweisungen“ auf Seite 20

Informationen zum Erstellen einer Initialisierungsdatei finden Sie in „Erstellen einer Initialisierungsdatei“ auf Seite 8.

Die in Abb. 2 gezeigte Beispielinitialisierungsdatei enthält Beispiele dieser Anweisungen.

1 DTW SMTP_SERVER 9.5.5.78	• Zeile 1 definiert die Werte der Konfigurationsvariablen.
2 MACRO_PATH /WWW/MACRO;/QSYS.LIB/WWW.LIB/MACRO.FILE	• Die Zeilen 2 - 4 definieren Pfade zu den Dateien, auf die Net.Data zugreifen muß.
3 INCLUDE_PATH /WWW/MACRO;/QSYS.LIB/WWW.LIB/MACRO.FILE	
4 EXEC_PATH /QSYS.LIB;/QSYS.LIB/WWW.LIB	
5 ENVIRONMENT(MYLE1) /QSYS.LIB/LELIB.LIB/MYLE1.SRVPGM (IN VAR1, OUT VAR2)	• Zeile 5 gibt eine benutzerdefinierte Anweisung ENVIRONMENT an.

Abbildung 2. Die Net.Data-Initialisierungsdatei

Der Text jeder einzelnen Konfigurationsanweisung muß vollständig in einer Zeile stehen. (Eine Anweisung ENVIRONMENT wird in mehreren Zeilen wiedergegeben, um die Lesbarkeit zu verbessern.) Stellen Sie sicher, daß die Initialisierungsdatei eine Anweisung ENVIRONMENT für jede benutzerdefinierte Sprachumgebung enthält, die Sie von Ihren Makros aus aufrufen. Sie müssen keine Pfadkonfigurationsanweisungen angeben, wenn Sie bei allen Verweisen auf Dateien innerhalb des Makros den vollständig qualifizierten Pfad angeben.

In den folgenden Abschnitten wird beschrieben, wie Sie die Initialisierungsdatei erstellen und die Konfigurationsanweisungen in der Initialisierungsdatei anpassen können.

Erstellen einer Initialisierungsdatei

Die Erstellung einer Initialisierungsdatei ist wahlfrei, wenn Sie Net.Data für OS/400 verwenden. Eine Initialisierungsdatei muß in folgenden Fällen erstellt werden:

- Sie möchten eine der Net.Data-Konfigurationsvariablen auf einen Wert setzen, der vom Standardwert abweicht.
- Sie möchten die Pfadanweisungen für Makro-, Kopf- und Programmdateien definieren, um Verweise auf diese Dateien zu verkürzen.
- Sie verwenden eine Sprachumgebung, die nicht in Net.Data enthalten ist.

Gehen Sie wie folgt vor, um eine Initialisierungsdatei zu erstellen:

1. Verwenden Sie in der Bibliothek, in der sich das Programmdateiobjekt DB2WWW befindet, den Befehl CRTSRCPF (Physische Quellendatei erstellen), um die Initialisierungsdatei zu erstellen.

Dateiname: INI

Member-Name: DB2WWW

Es empfiehlt sich, die Initialisierungsdatei mit der Satzlänge 240 zu erstellen, weil der Text der Konfigurationsanweisungen vollständig in einer Zeile stehen muß.

2. Verwenden Sie das Quelleneingabedienstprogramm (SEU) oder einen Workstation-Editor, um der Datei Konfigurationsanweisungen hinzuzufügen, wie im Beispielmakro und in den folgenden Abschnitten veranschaulicht.

Wenn Sie eine Initialisierungsdatei erstellen und sie dann aktualisieren, müssen Sie den Web-Server weder beenden noch erneut starten, damit die Änderungen wirksam werden. Net.Data liest die Initialisierungsdatei einmal, nämlich beim ersten Aufruf durch einen HTTP-Server-Job. Die Konfigurationsdaten werden gespeichert, so daß Net.Data die Initialisierungsdatei bei nachfolgenden Aufrufen von Net.Data nicht lesen muß. Wenn jedoch eine Änderung an der Initialisierungsdatei vorgenommen wird, erkennt Net.Data die Änderung an der Initialisierungsdatei und liest sie erneut.

Hinweis zu Berechtigungen: Stellen Sie sicher, daß die Benutzer-IDs, unter denen Net.Data ausgeführt wird, die entsprechenden Zugriffsrechte für diese Datei haben. Weitere Informationen hierzu finden Sie in „Erteilen von Zugriffsrechten für Objekte, auf die Net.Data zugreift“ auf Seite 25.

Konfigurationsvariablenanweisungen

Die Net.Data-Konfigurationsvariablenanweisungen legen die Werte der Konfigurationsvariablen fest. Konfigurationsvariablen werden für verschiedene Zwecke verwendet. Einige Variablen sind bei bestimmten Sprachumgebungen für die ordnungsgemäße Funktion oder den Betrieb in einem alternativen Modus erforderlich. Andere Variablen steuern die Zeichenverschlüsselung oder den Inhalt der momentan erstellten Web-Seite. Mit Konfigurationsvariablenanweisungen können Sie zudem anwendungsspezifische Variablen definieren.

Die verwendeten Konfigurationsvariablen hängen von den verwendeten Sprachumgebungen sowie anderen, anwendungsspezifischen Faktoren ab.

Gehen Sie wie folgt vor, um die Konfigurationsvariablenanweisungen zu aktualisieren:

Passen Sie die Initialisierungsdatei mit den Konfigurationsvariablen an, die für Ihre Anwendung erforderlich sind. Eine Konfigurationsvariable hat die folgende Syntax:

NAME[=]value-string

Das Gleichheitszeichen ist wahlfrei, was durch die eckigen Klammern angegeben wird.

In den folgenden Unterabschnitten werden die Konfigurationsvariablenanweisungen beschrieben, die Sie in der Initialisierungsdatei angeben können:

- „DTW_MACRO_CACHE_SIZE: Variable für Makro-Cache-Größe“
- „DTW_PAD_PGM_PARMS: Konfigurationsvariable für Parameterauffüllung“ auf Seite 10
- „DTW_SHOWSQL: Konfigurationsvariable zum Aktivieren bzw. Inaktivieren von SHOWSQL“ auf Seite 10
- „DTW_SMTP_CCSD: CCSID-Variable für E-Mail-SMTP“ auf Seite 11
- „DTW_SMTP_CHARSET: Variable für E-Mail-SMTP-Zeichensatz“ auf Seite 11
- „DTW_SMTP_SERVER: Variable für E-Mail-SMTP-Server“ auf Seite 12
- „DTW_SQL_ISOLATION: DB2-Isolationsvariable“ auf Seite 12
- „DTW_SQL_NAMING_MODE: SQL-Tabellenbenennungsvariable“ auf Seite 14
- „DTWR_CLOSE_REGISTRIES: Variable zum Öffnen der Web-Registrierdatenbank“ auf Seite 14

DTW_MACRO_CACHE_SIZE: Variable für Makro-Cache-Größe

Gibt die Speicherkapazität in Megabyte an, die Net.Data beim Zwischenspeichern von Makros verwenden soll. Wenn die Cache-Größe überschritten wird, entfernt Net.Data alte zwischengespeicherte Makros, um im Cache Speicherplatz freizugeben. Net.Data entfernt die Makros, deren Verwendung am weitesten zurückliegt.

Syntax:

DTW_MACRO_CACHE_SIZE [=] *size*

Dabei gilt folgendes:

size Gibt die Cache-Größe in Anzahl Megabyte an. Der Standardwert ist 5 MB, und Caching ist immer aktiviert. Wenn *size* 0 ist, werden keine Makros zwischengespeichert. Wenn *size* zwischen 1 - 4 liegt, wird der Standardwert 5 verwendet.

Beispiel: Angeben der Cache-Größe 16 MB

DTW_MACRO_CACHE_SIZE 16

DTW_PAD_PGM_PARMS: Konfigurationsvariable für Parameterauffüllung

Zeigt einer Sprachumgebung an, ob Zeichenparameter, die an ein Programm oder eine gespeicherte Prozedur übergeben werden sollen, mit Leerzeichen aufgefüllt werden. Zeichenparameter haben den Datentyp CHARACTER oder CHAR.

Wenn bei IN- oder INOUT-Parametern die Länge des Parameterwerts die angegebene Genauigkeit unterschreitet, werden rechts vom Parameterwert Leerzeichen eingefügt, bis die Länge des Parameterwerts mit der Genauigkeit übereinstimmt.

Bei OUT-Parametern wird der Parameterwert auf die durch *precision* angegebene Anzahl Leerzeichen gesetzt.

Nach dem Aufruf des Programms bzw. der gespeicherten Prozedur werden alle folgenden Leerzeichen aus den OUT- und INOUT-Parameterwerten entfernt.

Setzen Sie diese Variable in der Net.Data-Initialisierungsdatei auf einen Wert für alle Makros. Sie können den Wert durch eine entsprechende Definition im Makro überschreiben. Wenn DTW_PAD_PGM_PARMS nicht im Makro definiert ist, verwendet diese Variable den Wert in der Initialisierungsdatei.

DTW_PAD_PGM_PARMS wird von der Direktaufruf- und SQL-Sprachumgebung unterstützt.

Syntax:

DTW_PAD_PGM_PARMS [=] YES|NO

Dabei gilt folgendes:

- | | |
|------------|--|
| YES | Gibt an, daß alle IN- und INOUT-Zeichenparameterwerte linksbündig sind und auf die definierte Genauigkeit des Parameters mit Leerzeichen aufgefüllt werden, bevor die Parameter an das Programm bzw. die gespeicherte Prozedur übergeben werden. Folgende Leerzeichen werden nach dem Aufruf des Programms bzw. der gespeicherten Prozedur entfernt. |
| NO | Gibt an, daß Zeichenparameterwerte nicht mit Leerzeichen aufgefüllt werden (Werte enden auf Null), wenn Parameter an Programme bzw. gespeicherte Prozeduren übergeben werden. Folgende Leerzeichen werden nach dem Aufruf eines Programms bzw. einer gespeicherten Prozedur nicht entfernt. Dies ist der Standardwert. |

DTW_SHOWSQL: Konfigurationsvariable zum Aktivieren bzw. Inaktivieren von SHOWSQL

Überschreibt die Wirksamkeit der Einstellung SHOWSQL in Ihren Net.Data-Makros.

Syntax:

DTW_SHOWSQL YES|NO

Dabei gilt folgendes:

- | | |
|------------|---|
| YES | Aktiviert SHOWSQL in einem Makro, das den Wert von SHOWSQL auf YES setzt. |
| NO | Inaktiviert SHOWSQL in Ihren Makros, selbst wenn die Variable SHOWSQL auf YES gesetzt ist. Der Standardwert ist NO. |

Tabelle 1 beschreibt, wie die Einstellungen in der Net.Data-Initialisierungsdatei und das Makro festlegen, ob die Variable SHOWSQL für ein bestimmtes Makro aktiviert oder inaktiviert ist.

Tabelle 1. Die Beziehung zwischen den Einstellungen in der Net.Data-Initialisierungsdatei und dem Makro für SHOWSQL

Einstellung von DTW_SHOWSQL	Einstellung SHOWSQL	SQL-Anweisung angezeigt
NO	NO	NO
NO	YES	NO
YES	NO	NO
YES	YES	YES

DTW SMTP CCSID: CCSID-Variable für E-Mail-SMTP

Gibt die ASCII-CCSID (Coded Character Set Identifier - ID für codierten Zeichensatz) an, die dem in DTW SMTP_CHARSET angegebenen MIME-Zeichensatz (MIME - Multi-purpose Internet Mail Extensions) zugeordnet ist. Die CCSID muß verwendet werden, wenn in der Funktion DTW_SENDMAIL angegebene Daten von EBCDIC in ASCII umgesetzt werden.

Wird DTW SMTP_CCSID angegeben, müssen Sie auch DTW SMTP_CHARSET angeben. Stellen Sie bei Angabe der CCSID sicher, daß sie für den in DTW SMTP_CHARSET angegebenen MIME-Zeichensatz geeignet ist und daß die CCSID vom System unterstützt wird. Tabelle 2 auf Seite 12 listet gängige MIME-Zeichensätze und die zugeordnete ASCII-CCSID auf.

Wenn DTW SMTP_CCSID nicht festgelegt wird, verwendet Net.Data die CCSID, die dem MIME-Zeichensatz ISO-8859-1 zugeordnet ist, also 819.

Syntax:

DTW SMTP_CCSID [=] *ascii_ccsid*

Dabei gilt folgendes: *ascii_ccsid* ist die beim Umsetzen von EBCDIC in ASCII zu verwendende ASCII-CCSID (eine Zahl zwischen 1 und 65534).

Beispiel:

DTW SMTP_CCSID 912

Diese ASCII-CCSID entspricht dem MIME-Zeichensatz ISO-8859-2.

DTW SMTP_CHARSET: Variable für E-Mail-SMTP-Zeichensatz

Gibt den MIME-Zeichensatz (MIME - Multi-purpose Internet Mail Extensions) an, der in E-Mail-Nachrichten von der Funktion DTW_SENDMAIL verwendet werden soll. Wird DTW SMTP_CHARSET angegeben, müssen Sie auch DTW SMTP_CCSID angeben. Stellen Sie bei Angabe des MIME-Zeichensatzes sicher, daß der Zeichensatz gültig ist, denn Net.Data prüft den für diese Variable angegebenen Wert nicht. Wenn DTW SMTP_CHARSET nicht festgelegt wird, verwendet Net.Data den MIME-Zeichensatz ISO-8859-1 mit der zugeordneten CCSID 819.

Tabelle 2 listet gängige MIME-Zeichensätze und die zugeordnete ASCII-CCSID auf.

Tabelle 2. Von Net.Data unterstützte Zeichensätze

MIME-Standardzeichensatz	ASCII-CCSID	Beschreibung
US-ASCII	367	Amerikanisches Englisch
ISO-2022-JP	5052	MBCS (Japan)
ISO-8859-1	819	Lateinisch-1
ISO-8859-2	912	Lateinisch-2
ISO-8859-5	915	Kyrillisch
ISO-8859-6	1089	Arabisch
ISO-8859-7	813	Griechisch
ISO-8859-8	916	Hebräisch
ISO-8859-9	920	Lateinisch-5

Syntax:

`DTW SMTP_CHARSET character_set`

Dabei gilt folgendes: *character_set* ist der zu verwendende MIME-Zeichensatz.

Beispiel:

`DTW SMTP_CHARSET iso-8859-2`

Dieser MIME-Zeichensatz entspricht der ASCII-CCSID 912.

DTW SMTP_SERVER: Variable für E-Mail-SMTP-Server

Gibt den SMTP-Server an, der zum Senden von E-Mail-Nachrichten über die integrierte Funktion DTW_SENMAIL verwendet werden soll. Der Wert dieser Variablen kann ein Host-Name oder eine IP-Adresse sein. Wenn diese Variable nicht gesetzt wird, verwendet Net.Data den lokalen Host als SMTP-Server.

Syntax:

`DTW SMTP_SERVER server_name`

Dabei gilt folgendes: *server_name* ist der Host-Name bzw. die IP-Adresse des SMTP-Servers, der zum Senden von E-Mail-Nachrichten verwendet werden soll.

Hinweis zur Leistung: Geben Sie eine IP-Adresse für diesen Wert an, um zu verhindern, daß Net.Data die Verbindung zu einem Domännennamens-Server herstellt, wenn die IP-Adresse des angegebenen SMTP-Servers abgerufen wird.

Beispiel:

`DTW SMTP_SERVER 9.5.34.5`

DTW SQL_ISOLATION: DB2-Isolationsvariable

Die Sprachumgebung DTW_SQL verwendet die Konfigurationsanweisung DTW_SQL_ISOLATION, um festzulegen, in welchem Umfang Datenbankoperationen, die von der Sprachumgebung DTW_SQL ausgeführt werden, von gleichzeitig ablaufenden Prozessen isoliert werden.

Syntax:

DTW_SQL_ISOLATION *locking_method*

Dabei gilt folgendes: *locking_method* ist einer der folgenden Werte:

DTW_SQL_NO_COMMIT

Gibt an, daß die COMMIT-Steuerung nicht verwendet wird. Geben Sie diesen Wert beim Betriebssystem OS/400 nicht an, wenn im Verzeichnis für die relationale Datenbank eine relationale Datenbank angegeben wurde und sich diese auf einem System befindet, das nicht unter OS/400 ausgeführt wird.

DTW_SQL_READ_UNCOMMITTED

Gibt die Sperrung für die Objekte an, auf die in den Anweisungen SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON und REVOKE verwiesen wird, sowie für die aktualisierten, gelöschten und eingefügten Zeilen. Die Objekte werden bis zum Ende der Arbeitseinheit (Transaktion) gesperrt. Nicht festgeschriebene Änderungen in anderen Prozessen sind sichtbar.

DTW_SQL_READ_COMMITTED

Gibt die Sperrung für die Objekte an, auf die in den Anweisungen SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON und REVOKE verwiesen wird, sowie für die aktualisierten, gelöschten und eingefügten Zeilen. Die Objekte werden bis zum Ende der Arbeitseinheit (Transaktion) gesperrt. Eine Zeile, die ausgewählt ist, jedoch nicht aktualisiert wird, wird gesperrt, bis die nächste Zeile ausgewählt wird. Nicht festgeschriebene Änderungen in anderen Prozessen sind nicht sichtbar.

DTW_SQL_REPEATABLE_READ

Gibt die Sperrung für die Objekte an, auf die in den Anweisungen SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON und REVOKE verwiesen wird, sowie für die ausgewählten, aktualisierten, gelöschten und eingefügten Zeilen. Die Objekte werden bis zum Ende der Arbeitseinheit (Transaktion) gesperrt. Nicht festgeschriebene Änderungen in anderen Prozessen sind nicht sichtbar.

DTW_SQL_SERIALIZABLE

Gibt die Sperrung für die Objekte an, auf die in den Anweisungen SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON und REVOKE verwiesen wird, sowie für die ausgewählten, aktualisierten, gelöschten und eingefügten Zeilen. Die Objekte werden bis zum Ende der Arbeitseinheit (Transaktion) gesperrt. Nicht festgeschriebene Änderungen in anderen Prozessen sind nicht sichtbar. Alle Tabellen, auf die in den Anweisungen SELECT, UPDATE, DELETE und INSERT verwiesen wird, werden bis zum Ende der Arbeitseinheit (Transaktion) zur ausschließlichen Nutzung gesperrt.

DTW_SQL_NAMING_MODE: SQL-Tabellenbenennungsvariable

Die Konfigurationsanweisung DTW_SQL_NAMING_MODE legt fest, wie ein Tabellenname in einer SQL-Anweisung angegeben werden kann.

Syntax:

DTW_SQL_NAMING_MODE *mode*

Dabei gilt folgendes: *mode* ist einer der folgenden Werte:

SQL_NAMING

Gibt an, daß Tabellen im folgenden Format durch den Datensammlungsnamen qualifiziert werden:

collection.table

Hierbei ist *collection* der Name der Datensammlung und *table* der Tabellenname. Das Standardqualifikationsmerkmal ist die Benutzer-ID, die den Prozeß ausführt, der wiederum die SQL-Anweisung ausführt; es wird verwendet, wenn der Tabellenname nicht explizit qualifiziert und der Standarddatensammlungsname nicht angegeben wird. SQL_NAMING ist der Standardtabellenname.

SYSTEM_NAMING

Gibt an, daß Dateien im folgenden Format durch den Bibliotheksnamen qualifiziert werden:

library/file

Hierbei ist *library* der Name der Bibliothek und *file* der Tabellenname. Der Standardsuchpfad ist die Bibliotheksliste (*LIBL) für den Tabellennamen ohne Qualifikationsmerkmal, wenn der Tabellenname (*file*) nicht explizit qualifiziert und kein Standarddatensammlungsname (*library*) angegeben wird.

DTWR_CLOSE_REGISTRIES: Variable zum Öffnen der Web-Registrierdatenbank

Gibt an, ob eine Web-Registrierdatenbank geschlossen oder offen gehalten werden soll. Mit dieser Variable kann die Web-Registrierdatenbank offen gehalten werden, so daß nachfolgende Aufrufe von Net.Data-Makros, die auf die gleiche Registrierdatenbank zugreifen, die Registrierdatenbank nicht erneut öffnen müssen.

Syntax:

DTWR_CLOSE_REGISTRIES YES|NO

Dabei gilt folgendes:

YES Gibt an, daß alle offenen Web-Registrierdatenbanken nach der Verarbeitung eines Net.Data-Makros geschlossen werden.

NO Gibt an, daß alle offenen Web-Registrierdatenbanken nach der Verarbeitung eines Net.Data-Makros offen gehalten werden. Der Standardwert ist NO.

Hinweis zur Leistung: Sie können die Konfigurationsanweisung DTWR_CLOSE_REGISTRIES verwenden, um die Leistung beim Zugriff auf eine Web-Registrierdatenbank (mit den integrierten Funktionen der Web-Registrierdatenbank) zu erhöhen, indem Registrierdatenbanken weniger oft geöffnet und geschlossen werden müssen.

Wenn mehrere Prozesse gleichzeitig auf die Registrierdatenbank zugreifen können (wie dies bei simultaner Browser-Anforderung der Fall ist), setzen Sie DTWR_CLOSE_REGISTRIES auf YES.

Pfadkonfigurationsanweisungen

Net.Data ermittelt die Speicherposition der Dateien und ausführbaren Programme, die von Net.Data-Makros verwendet werden, anhand der Einstellungen der Pfadkonfigurationsanweisungen. Es gibt folgende Pfadanweisungen:

- „MACRO_PATH“ auf Seite 16
- „EXEC_PATH“ auf Seite 17
- „INCLUDE_PATH“ auf Seite 18
- „FFI_PATH“ auf Seite 19
- „HTML_PATH“ auf Seite 19
- „DTW_JAVA_CLASSPATH“ auf Seite 20

Diese Pfadanweisungen geben mindestens ein Verzeichnis an, das Net.Data durchsucht, wenn es versucht, Makros, ausführbare Dateien, Textdateien, und Kopfdateien zu lokalisieren. Die benötigten Pfadanweisungen hängen von dem Net.Data-Leistungsspektrum ab, das Ihre Makros verwenden.

Aktualisierungsrichtlinien:

Für die Pfadanweisungen gelten mehrere allgemeine Richtlinien. Ausnahmen werden in den Beschreibungen der einzelnen Pfadanweisungen angemerkt.

- Jedes angegebene Verzeichnis wird durch ein Semikolon (;) beendet.
- Schrägstriche (/) und umgekehrte Schrägstriche (\) werden auf gleiche Weise behandelt.
- Jede Pfadanweisung kann mehrere Pfade angeben. Pfade werden von links nach rechts in der angegebenen Reihenfolge durchsucht. Durch die Möglichkeit zur Angabe mehrerer Pfade können Sie Ihre Dateien in mehreren Verzeichnissen verwalten. Sie können zum Beispiel jede Ihrer Web-Anwendungen in einem separaten Verzeichnis ablegen.
- Es wird empfohlen, absolute Pfadanweisungen zu verwenden.

In den folgenden Abschnitten werden der Zweck und die Syntax jeder Pfadanweisung beschrieben und Beispiele gültiger Pfadanweisungen gegeben.

MACRO_PATH

Die Konfigurationsanweisung MACRO_PATH gibt die Verzeichnisse an, die Net.Data nach Net.Data-Makros durchsucht. Zum Beispiel wird durch Angabe der folgenden URL-Adresse das Net.Data-Makro mit dem Pfad und Dateinamen /macro/sqlm.d2w angefordert:

```
http://server/cgi-bin/db2www/macro/sqlm.d2w/report
```

Syntax:

```
MACRO_PATH [=] path1;path2;...;pathn
```

Das Gleichheitszeichen (=) ist wahlfrei, wie durch eckige Klammern angegeben.

Net.Data fügt den Pfad /macro/sqlm.d2w an die Pfade in der Konfigurationsanweisung MACRO_PATH von links nach rechts an, bis das Net.Data-Makro gefunden wird bzw. alle Pfade durchsucht worden sind. Informationen zum Aufrufen von Net.Data-Makros finden Sie in „Aufrufen von Net.Data“ auf Seite 37.

Beispiel: Das folgende Beispiel zeigt die Anweisung MACRO_PATH in der Initialisierungsdatei und die zugehörige Programmverbindung (Link), die Net.Data aufruft.

Net.Data-Initialisierungsdatei:

```
MACRO_PATH /u/user1/macros;/usr/lpp/netdata/macros;
```

HTML-Programmverbindung (Link):

```
<A HREF="http://server/cgi-bin/db2www/query.d2w/input">Submit  
another query.</A>
```

Wenn die Datei query.d2w im Verzeichnis /u/user1/macros gefunden wird, ist der vollständig qualifizierte Pfad /u/user1/macros/query.d2w.

Wenn die Datei in den in der Anweisung MACRO_PATH angegebenen Verzeichnissen nicht gefunden wird, sucht Net.Data die Datei im Stammverzeichnis (/). Beispiel: Die folgende URL-Adresse wird übergeben:

```
http://myserver/cgi-bin/db2www/myfile.txt/report
```

Die Datei myfile.txt wurde in keinem der in MACRO_PATH angegebenen Verzeichnisse gefunden. Net.Data versucht dann, die Datei im Stammverzeichnis (/) zu finden:

```
/myfile.txt
```

EXEC_PATH

Die Konfigurationsanweisung EXEC_PATH gibt ein oder mehrere Verzeichnisse an, die Net.Data nach einem externen Programm durchsucht, das über die EXEC-Anweisung oder eine ausführbare Variable aufgerufen wird. Wenn das Programm gefunden wird, wird der Name des externen Programms an die Pfadangabe angefügt. Der daraus resultierende, vollständig qualifizierte Dateiname wird zur Ausführung an die Sprachumgebung übergeben.

Syntax:

```
EXEC_PATH [=] path1;path2;...;pathn
```

Beispiel: Das folgende Beispiel zeigt die Anweisung EXEC_PATH in der Initialisierungsdatei und die Anweisung EXEC im Makro, das das externe Programm aufruft.

Net.Data-Initialisierungsdatei:

```
EXEC_PATH /qsys.lib/programs.lib;/qsys.lib/rexx.lib/rexxpgms.file;
```

Net.Data-Makro:

```
%FUNCTION(DTW_REXX) myFunction() {  
    %EXEC{ myFunction.mbr %}  
%}
```

Wenn die Datei MyFunction.mbr im Verzeichnis /qsys.lib/rexx.lib/rexxpgms.file gefunden wird, ist der qualifizierte Name des Programms /qsys.lib/rexx.lib/rexxpgms.file/myFunction.mbr.

Wenn die Datei in den in der Anweisung EXEC_PATH angegebenen Verzeichnissen nicht gefunden wird, gilt folgendes:

- Wenn der angegebene Pfad absolut ist, sucht Net.Data die Datei im angegebenen Pfad. Beispiel: Die folgende EXEC-Anweisung wurde angegeben:

```
%EXEC{/qsys.lib/programs.lib/rpg1.pgm %}
```

Net.Data sucht dann die Datei rpg1.pgm im Verzeichnis /qsys.lib/programs.lib.

- Wenn der angegebene Pfad relativ ist, sucht Net.Data die Datei im aktuellen Arbeitsverzeichnis. Beispiel: Die folgende EXEC-Anweisung wurde angegeben:

```
%EXEC { rpg1.pgm %}
```

Net.Data versucht dann, die Datei rpg1.pgm im aktuellen Arbeitsverzeichnis zu finden.

INCLUDE_PATH

Die Konfigurationsanweisung INCLUDE_PATH gibt ein oder mehrere Verzeichnisse an, die Net.Data durchsucht, um eine in einer INCLUDE-Anweisung in einem Net.Data-Makro angegebene Datei zu finden. Wenn Net.Data die Datei findet, hängt es den Namen der Kopffdatei an die Pfadangabe an, um den qualifizierten Kopffdateinamen zu erstellen.

Syntax:

```
INCLUDE_PATH [=] path1;path2;...;pathn
```

Beispiel 1: Das folgende Beispiel zeigt sowohl die Anweisung INCLUDE_PATH in der Initialisierungsdatei als auch die Anweisung INCLUDE, die die Kopffdatei angibt.

Net.Data-Initialisierungsdatei:

```
INCLUDE_PATH /u/user1/includes;/usr/lpp/netdata/includes;
```

Net.Data-Makro:

```
%INCLUDE "myInclude.txt"
```

Wenn die Datei myInclude.txt im Verzeichnis /u/user1/includes gefunden wird, ist der vollständig qualifizierte Name der Kopffdatei /u/user1/includes/myInclude.txt.

Beispiel 2: Das folgende Beispiel zeigt die Anweisung INCLUDE_PATH und eine Kopffdatei mit einem Unterverzeichnisnamen.

Net.Data-Initialisierungsdatei:

```
INCLUDE_PATH /u/user1/includes;/usr/lpp/netdata/includes;
```

Net.Data-Makro:

```
%INCLUDE "OE/oeheader.inc"
```

Die Kopffdatei wird in den Verzeichnissen /u/user1/includes/OE und /usr/lpp/netdata/includes/OE gesucht. Wenn die Datei im Verzeichnis /usr/lpp/netdata/includes/OE gefunden wird, ist der vollständig qualifizierte Name der Kopffdatei /usr/lpp/netdata/includes/OE/oeheader.inc.

Wenn die Datei in den in der Anweisung INCLUDE_PATH angegebenen Verzeichnissen nicht gefunden wird, gilt folgendes:

- Wenn der angegebene Pfad absolut ist, sucht Net.Data die Datei im angegebenen Pfad. Beispiel: Die folgende INCLUDE-Anweisung wurde angegeben:

```
%INCLUDE "/u/user1/includes/oeheader.inc"
```

Net.Data sucht dann die Datei oeheader.inc im Verzeichnis /u/user1/includes.

- Wenn der angegebene Pfad relativ ist, sucht Net.Data die Datei im aktuellen Arbeitsverzeichnis. Beispiel: Die folgende INCLUDE-Anweisung wurde angegeben:

```
%INCLUDE "oeheader.inc"
```

Net.Data versucht dann, die Datei oeheader.inc im aktuellen Arbeitsverzeichnis zu finden.

FFI_PATH

Die Konfigurationsanweisung FFI_PATH gibt ein oder mehrere Verzeichnisse an, in denen Net.Data nach einer unstrukturierten Textdatei sucht, auf die durch die FFI-Funktion (FFI - Flat File Interface - Schnittstelle für unstrukturierte Dateien) verwiesen wird.

Syntax:

```
FFI_PATH [=] path1;path2;...;pathn
```

Beispiel: Das folgende Beispiel zeigt eine Anweisung FFI_PATH in der Initialisierungsdatei.

Net.Data-Initialisierungsdatei:

```
FFI_PATH /u/user1/ffi;/usr/lpp/netdata/ffi;
```

Wenn die FFI-Sprachumgebung aufgerufen wird, sucht Net.Data im in der Anweisung FFI_PATH angegebenen Pfad.

Da die Anweisung FFI_PATH verwendet wird, um die nicht in der Pfadanweisung aufgeführten Dateien zu sichern, gelten für nicht gefundene FFI-Dateien besondere Regeln. Weitere Informationen finden Sie im Abschnitt über in FFI integrierte Funktionen im Handbuch *Net.Data Reference*.

HTML_PATH

Die Konfigurationsanweisung HTML_PATH gibt an, in welches Verzeichnis Net.Data große Objekte (LOBs) schreibt. Für diese Pfadanweisung ist nur ein Verzeichnispfad zulässig.

Die Anweisung HTML_PATH muß ein IFS-Verzeichnis angeben, das sich nicht im Dateisystem QSYS.LIB befindet.

Syntax:

```
HTML_PATH [=] path
```

Beispiel: Das folgende Beispiel zeigt die Anweisung HTML_PATH in der Initialisierungsdatei.

Net.Data-Initialisierungsdatei:

```
HTML_PATH /db2/lobs
```

Wenn eine Abfrage ein LOB zurückgibt, sichert Net.Data es in dem in der in der Konfigurationsanweisung HTML_PATH angegebenen Verzeichnis.

Hinweis zur Leistung: Berücksichtigen Sie bei der Verwendung von LOBs die Systemeinschränkungen, denn LOBs können Ressourcen schnell aufbrauchen. Weitere Informationen finden Sie in „Verwenden großer Objekte“ auf Seite 105.

DTW_JAVA_CLASSPATH

Die Konfigurationsanweisung DTW_JAVA_CLASSPATH gibt den Pfad zum Lokalisieren von Java-Klassen an. Verzeichnisse werden durch Doppelpunkte voneinander getrennt.

Syntax:

DTW_JAVA_CLASSPATH [=] *path*

Beispiel: Das folgende Beispiel zeigt die Anweisung DTW_JAVA_CLASSPATH in der Initialisierungsdatei.

Net.Data-Initialisierungsdatei:

```
DTW_JAVA_CLASSPATH /directory1/directory2:/QIBM/ProdData/Java400
```

Umgebungskonfigurationsanweisungen

Eine Anweisung ENVIRONMENT konfiguriert eine Sprachumgebung. Eine Sprachumgebung ist eine Net.Data-Komponente, mit der Net.Data auf eine Datenquelle, wie zum Beispiel eine DB2-Datenbank, zugreift oder ein in einer Sprache wie REXX geschriebenes Programm ausführt. Net.Data stellt eine Reihe von Sprachumgebungen bereit sowie eine Schnittstelle, mit der Sie Ihre eigenen Sprachumgebungen erstellen können. Diese Sprachumgebungen werden in „Verwenden der Sprachumgebungen“ auf Seite 91 beschrieben, und die Schnittstelle für Sprachumgebungen wird im Handbuch *Net.Data Language Environment Interface Reference* erörtert.

Net.Data erfordert, daß eine Anweisung ENVIRONMENT für eine bestimmte Sprachumgebung vorhanden ist, bevor Sie diese Sprachumgebung aufrufen können.

Net.Data für OS/400 erfordert keine Anweisung ENVIRONMENT für die mit Net.Data ausgelieferten Sprachumgebungen. Wenn jedoch eine Sprachumgebungsanweisung festgestellt wird, überschreibt sie den von Net.Data verwendeten Standardwert. Es wird empfohlen, daß die Anweisungen ENVIRONMENT für die mit Net.Data ausgelieferten Sprachumgebungen der Net.Data -Konfigurationsdatei nicht hinzugefügt werden.

Sie können Variablen einer Sprachumgebung zuordnen, indem Sie die Variablen als Parameter in der Anweisung ENVIRONMENT angeben. Net.Data übergibt die in einer Anweisung ENVIRONMENT angegebenen Parameter implizit als Makrovariablen an die Sprachumgebung. Sie können im Makro den Wert eines Parameters ändern, der in einer Anweisung ENVIRONMENT angegeben ist, indem Sie entweder der Variablen mit der Funktion DTW_ASSIGN() einen Wert zuordnen oder die Variable in einem DEFINE-Abschnitt definieren.

Wichtig: Wird eine Makrovariable in einem Makro definiert, aber nicht in der Anweisung ENVIRONMENT angegeben, wird die Makrovariable nicht an die Sprachumgebung übergeben.

Zum Beispiel kann ein Makro eine Variable DATABASE definieren, um den Namen einer Datenbank anzugeben, in der eine SQL-Anweisung innerhalb der Funktion DTW_SQL ausgeführt werden soll. Der Wert von DATABASE muß an die SQL-Sprachumgebung (DTW_SQL) übergeben werden, damit die SQL- br Sprachumgebung die Verbindung zur angegebenen Datenbank herstellen kann. Zum Übergeben der Variablen an die Sprachumgebung müssen Sie die Variable DATABASE der Parameterliste in der Umgebungsanweisung für DTW_SQL hinzufügen.

Die Net.Data-Beispielinitialisierungsdatei geht beim Anpassen der Einstellungen der Anweisungen für die Net.Data-Umgebungsconfiguration von bestimmten Annahmen aus. Diese Annahmen sind für Ihre Umgebung eventuell nicht zutreffend. Ändern Sie die Anweisungen Ihrer Umgebung entsprechend.

Gehen Sie wie folgt vor, um eine Anweisung ENVIRONMENT hinzuzufügen oder zu aktualisieren:

Anweisungen ENVIRONMENT haben die folgende Syntax:

`ENVIRONMENT(type) library_name (parameter_list, ...)`

Parameter:

- *type*

Der Name, durch den Net.Data diese Sprachumgebung einem FUNCTION-Block zuordnet, der in einem Net.Data-Makro definiert ist. Sie müssen die Art der Sprachumgebung in einer FUNCTION-Blockdefinition angeben, um die Sprachumgebung zu definieren, in der Net.Data die Funktion ausführen soll.

- *library_name*

Der Name des Serviceprogramms mit den Schnittstellen für Sprachumgebungen, die Net.Data aufruft.

Der Name des Serviceprogramms wird mit der Erweiterung *.SRVPGM* angegeben.

- *parameter_list*

Die Liste der Parameter, die bei jedem Funktionsaufruf neben den Parametern, die in der FUNCTION-Blockdefinition angegeben sind, an die Sprachumgebung übergeben werden.

Definieren Sie die Variable im Makro, um die Variablen in der Parameterliste festzulegen und zu übergeben.

Sie müssen diese Parameter als Konfigurationsvariablen oder als Variablen in Ihrem Makro definieren, bevor Sie eine Funktion ausführen können, die von der Sprachumgebung verarbeitet wird. Wenn eine Funktion ihre Ausgabeparameter ändert, behalten die Parameter ihre geänderten Werte nach der Beendigung der Funktion bei.

Wenn Net.Data die Initialisierungsdatei verarbeitet, werden die Serviceprogramme der Sprachumgebung nicht geladen. Net.Data lädt ein Serviceprogramm, wenn es das erste Mal eine Funktion ausführt, die die betreffende Sprachumgebung angibt. Das Serviceprogramm bleibt dann so lange geladen, wie Net.Data geladen ist.

Beispiel: Anweisungen ENVIRONMENT für von Net.Data bereitgestellte Sprachumgebungen

Beim Anpassen der Anweisungen ENVIRONMENT für Ihre Anwendung müssen Sie den Anweisungen ENVIRONMENT die Variablen hinzufügen, die von Ihrer Initialisierungsdatei an die Sprachumgebung übergeben werden müssen, oder die Net.Data-Makroprogrammierer zum Festlegen oder Überschreiben in ihren Makros benötigen.

Unter OS/400 sind Anweisungen ENVIRONMENT für Net.Data-Sprachumgebungen nicht erforderlich und werden nicht empfohlen. Das folgende Beispiel zeigt jedoch einige der Standardanweisungen ENVIRONMENT, die Net.Data verwendet.

```
1  MACRO_PATH      /WWW/MACRO;/QSYS.LIB/WWW.LIB/MACRO.FILE
2  INCLUDE_PATH    /WWW/MACRO;/QSYS.LIB/WWW.LIB/MACRO.FILE
3  EXEC_PATH       /QSYS.LIB;/QSYS.LIB/WWW.LIB

4  ENVIRONMENT(DTW_REXX) /QSYS.LIB//QTCP.LIB/QTMHREXX.SRVPGM ( )
5  ENVIRONMENT(DTW_SQL)  /QSYS.LIB/QTCP.LIB/QTMSQL.SRVPGM (IN DATABASE,
    LOGIN, PASSWORD, TRANSACTION_SCOPE, SHOWSQL, DB_CASE,
    RPT_MAX_ROWS, START_ROW_NUM, DTW_SET_TOTAL_ROWS,
    OUT_DTWTABLE, SQL_CODE, TOTAL_ROWS)
6  ENVIRONMENT(DTW_SYSTEM) /QSYS.LIB/QTCP.LIB/QTMSYS.SRVPGM ( )
```

Wichtiger Hinweis: Jede Anweisung ENVIRONMENT muß in einer separaten Zeile stehen.

Einrichten der Sprachumgebungen

Nach einer Änderung von Konfigurationsvariablen und Konfigurationsanweisungen ENVIRONMENT für die Net.Data-Sprachumgebungen sind einige zusätzliche Konfigurationsschritte erforderlich, damit die folgenden Sprachumgebungen ordnungsgemäß funktionieren. In den folgenden Abschnitten werden die zum Definieren der Sprachumgebungen erforderlichen Schritte beschrieben:

- „Definieren der Java-Anwendungssprachumgebung“
- „Definieren der SQL-Sprachumgebung“ auf Seite 23

Definieren der Java-Anwendungssprachumgebung

Bevor Sie die Java-Anwendungssprachumgebung verwenden, die zuerst in OS/400 Version 4 Release 4 eingeführt wurde, führen Sie die folgenden Schritte aus:

1. Installieren Sie das Lizenzprogramm „AS/400 Developer Kit for Java“ Produktkennung 5769JV1. „AS/400 Developer Kit for Java“ muß installiert sein, um Java-Anwendungen auf dem System IBM AS/400 ausführen zu können.
2. Legen Sie die Pfadkonfigurationsvariable DTW_JAVA_CLASSPATH in der Net.Data-Initialisierungsdatei so fest, daß Java die Java-Anwendungsklassen finden kann. Weitere Informationen zu dieser Pfadkonfigurationsvariablen finden Sie in „DTW_JAVA_CLASSPATH“ auf Seite 20.

Informationen zur Verwendung der Java-Anwendungssprachumgebung nach ihrer Definition finden Sie in „Java-Anwendungssprachumgebung“ auf Seite 98.

Definieren der SQL-Sprachumgebung

Führen Sie vor der Verwendung der SQL-Sprachumgebung die folgenden Schritte aus:

1. Erstellen Sie einen Verzeichniseintrag für die lokale Datenbank im Verzeichnis für die relationale Datenbank (d. h. einen Verzeichniseintrag mit einer fernen Position *LOCAL). Erstellen Sie diesen Eintrag zusätzlich zu den Einträgen aller fernen Datenbanken, auf die die SQL-Sprachumgebung zugreifen muß.

Fügen Sie den Eintrag mit dem Befehl ADDRDBDIRE (RDB-Verzeichniseintrag hinzufügen) hinzu.

Wenn Sie auf eine ferne Datenbank zugreifen, führen Sie zusätzliche Konfigurationsschritte wie das Einrichten der Datenfernverarbeitung zwischen dem lokalen System und dem fernen System aus. Weitere Informationen zur Unterstützung verteilter Datenbanken finden Sie im Handbuch *OS/400 Distributed Database Programming*.
2. Wenn Sie DataLink-Verweise verwenden, stellen Sie sicher, daß TCP/IP auf allen verwendeten Systemen konfiguriert ist und daß DataLink File Manager auf allen Systemen, die zu verbindende Objekte enthalten werden, gestartet und konfiguriert ist. Weitere Informationen zu DataLinks finden Sie im Handbuch *DB2 for OS/400 SQL Programming*.
3. Wenn große Objekte (LOBs) von der SQL-Sprachumgebung zurückgegeben werden, legen Sie die Konfigurationsvariable HTML_PATH fest. Weitere Informationen zu dieser Konfigurationsvariablen finden Sie in „HTML_PATH“ auf Seite 19.
4. Fügen Sie Konfigurationsvariablen hinzu, oder aktualisieren Sie diese Variablen. Die SQL-Sprachumgebung unterstützt die folgenden Konfigurationsvariablen, die in einer Net.Data-Initialisierungsdatei angegeben werden können:

DTW_SQL_ISOLATION

Legt fest, in welchem Umfang Datenbankoperationen, die von der SQL-Sprachumgebung ausgeführt werden, von gleichzeitig ablaufenden Prozessen isoliert werden.

DTW_SQL_NAMING_MODE

Legt fest, wie ein Tabellenname in einer SQL-Anweisung angegeben werden kann.

DTW_SHOWSQL

Aktiviert die Verwendung der Makrovariablen SHOWSQL.

Weitere Informationen zu den Net.Data-Konfigurationsvariablenanweisungen finden Sie in „Konfigurationsvariablenanweisungen“ auf Seite 8.

Informationen zur Verwendung der SQL-Sprachumgebung nach Ihrer Definition finden Sie in „SQL-Sprachumgebung“ auf Seite 102.

Konfigurieren des Web-Servers

Common Gateway Interface (CGI) ist eine Standardschnittstelle, über die ein Web-Server ein Anwendungsprogramm wie Net.Data aufrufen kann. Dadurch, daß Net.Data CGI unterstützt, können Sie es mit Ihrem bevorzugten Web-Server verwenden.

Konfigurieren Sie den Web-Server so, daß Net.Data aufgerufen wird. Nehmen Sie dazu Map-, Exec- und Pass-Anweisungen in die HTTP-Konfigurationsdatei auf, die bewirken, daß Net.Data aufgerufen wird.

Wenn sich beispielsweise das Net.Data-Programmdateiobjekt in der Bibliothek CGI befindet, werden mit den folgenden Anweisungen Net.Data-Anforderungen nach /QSYS.LIB/CGI.LIB/DB2WWW.PGM umgeleitet:

```
Map /cgi-bin/db2www/* /QSYS.LIB/CGI.LIB/DB2WWW.PGM/*
Map /CGI-BIN/DB2WWW/* /QSYS.LIB/CGI.LIB/DB2WWW.PGM/*
Exec /QSYS.LIB/CGI.LIB/*
```

Empfehlung: Verwalten Sie die Anweisungen innerhalb der HTTP-Konfigurationsdatei in der folgenden Reihenfolge, um zu verhindern, daß Anweisungen ignoriert werden: Map, Exec, Pass. Wenn z. B. die folgende Pass-Anweisung einer Map- oder Exec-Anweisung vorangeht, werden die Map- und Exec-Anweisungen ignoriert:

```
Pass /*
```

Map-Anweisungen

Mit den Map-Anweisungen werden Einträge mit dem Format /cgi-bin/db2www/* der Bibliothek zugeordnet, in der sich das Net.Data-Programm auf Ihrem System befindet. (Der Stern (*) am Ende der Zeichenfolge bezieht sich auf alles, was auf die Zeichenfolge folgt.) Es werden sowohl Map-Anweisungen in Kleinbuchstaben als auch solche in Großbuchstaben aufgeführt, da bei den Anweisungen zwischen Groß-/Kleinschreibung unterschieden wird.

In diesem Beispiel verweisen beide Map-Anweisungen auf die gleiche Speicherposition.

Exec-Anweisungen

Mit der Exec-Anweisung können vom Web-Server beliebige CGI-Programme in der CGI-Bibliothek ausgeführt werden. Geben Sie die Bibliothek, in der sich das Programm befindet (nicht das Programm selbst), in der Anweisung an.

Pass-Anweisungen

Wenn Sie große Objekte (LOBs) in der SQL-Sprachumgebung verwenden wollen, erstellen Sie eine Pass-Anweisung, um das Verzeichnis anzugeben, in dem die SQL-Sprachumgebung die LOB-Dateien speichert. Beispiel:

```
Pass /tmplobs/* /html_path/*
```

Dabei gilt folgendes: *html_path* ist der in der Konfigurationsvariablen HTML_PATH angegebene Verzeichnisname, der das Standardverzeichnis zum Speichern von LOBs angibt. Weitere Informationen hierzu finden Sie in „HTML_PATH“ auf Seite 19.

Pass-Anweisungen werden von Net.Data nicht verwendet. Wenn Sie Ihre URL-Adresse vereinfachen möchten, können Sie die Anweisung MACRO_PATH in einer Net.Data-Initialisierungsdatei verwenden (siehe dazu „MACRO_PATH“ auf Seite 16).

Erteilen von Zugriffsrechten für Objekte, auf die Net.Data zugreift

Vor der Verwendung von Net.Data müssen Sie sicherstellen, daß die Benutzer-IDs, unter denen Net.Data ausgeführt wird, über die notwendigen Zugriffsrechte für die Objekte verfügen, auf die in einem Net.Data-Makro verwiesen wird. Diese Rechte werden auch für das Makro benötigt, auf das in einer URL-Adresse verwiesen wird.

Stellen Sie vor allem sicher, daß die Benutzer-IDs, unter denen Net.Data ausgeführt wird, über die folgenden Berechtigungen verfügen:

- Lesen der Net.Data-Initialisierungsdatei `INI.FILE/DB2WWW.MBR`
- Ausführen der ausführbaren Net.Data-Dateien und Serviceprogramme und Durchsuchen der Verzeichnisse (Bibliotheken) in den Pfaden zu den ausführbaren Dateien und Serviceprogrammen
- Lesen der entsprechenden Net.Data-Makrodateien und Durchsuchen der entsprechenden Verzeichnisse, die durch die Pfadkonfigurationsanweisung `MACRO_PATH` angegeben werden
- Ausführen der entsprechenden Dateien und Durchsuchen der entsprechenden Verzeichnisse, die durch die Pfadkonfigurationsanweisung `EXEC_PATH` angegeben werden
- Lesen der entsprechenden Dateien und Durchsuchen der entsprechenden Verzeichnisse, die durch die Pfadkonfigurationsanweisung `INCLUDE_PATH` angegeben werden
- Lesen und Schreiben der entsprechenden Dateien und Durchsuchen der entsprechenden Verzeichnisse, die durch die Pfadkonfigurationsanweisung `FFI_PATH` angegeben werden
- Zugreifen auf ein Objekt, auf das eventuell durch das Ziel einer Sprachumgebungsanweisung verwiesen wird. Die SQL-Sprachumgebung z. B. führt SQL-Anweisungen aus, und SQL-Anweisungen greifen auf Datenbankdateien zu. Das heißt, daß die Benutzer-ID, unter der Net.Data ausgeführt wird, über die Berechtigung für die Datenbankdateien verfügen muß.

Beispiele:

Je nach Dateisystem, in dem Sie Ihre Net.Data-Makros ablegen, müssen Sie dem Benutzerprofil, unter dem das Net.Data-CGI-Programm ausgeführt wird, eine Berechtigung für das Net.Data-Makro erteilen. Mit den folgenden Methoden wird dem Benutzerprofil QTMHHTTP1 eine Berechtigung erteilt (in V3R2 und V3R7 hat Internet Connection für AS/400 CGI-Programme nur unter dem Benutzerprofil QTMHHTTP1 ausgeführt). :

- Verwenden Sie im Stammdateisystem den CL-Befehl CHGAUT (Berechtigung ändern), um dem Benutzerprofil eine Berechtigung zu erteilen:

```
CHGAUT OBJ('/WWW') USER(QTMHHTTP1) DTAAUT(*RX)
CHGAUT OBJ('/WWW/macro') USER(QTMHHTTP1) DTAAUT(*RX)
CHGAUT OBJ('/WWW/macro/*') USER(QTMHHTTP1) DTAAUT(*RX)
```

Sie müssen allen Objekten im Pfad eine Berechtigung erteilen.

- Verwenden Sie im Bibliotheksdateisystem (QSYS.LIB) den CL-Befehl GRTOBJAUT (Objektberechtigung erteilen), um dem Benutzerprofil eine Berechtigung zu erteilen:

```
GRTOBJAUT OBJ(WWW) OBJTYPE(*LIB) USER(QTMHHTP1) AUT(*USE)
GRTOBJAUT OBJ(WWW/MACRO) OBJTYPE(*FILE) USER(QTMHHTP1) AUT(*USE)
```

Sie müssen nur der Bibliothek und der physischen Quellendatei eine Berechtigung erteilen.

Sie können auch den CL-Befehl CHGAUT wie folgt verwenden, um Objekten im Dateisystem QSYS.LIB eine Berechtigung zu erteilen:

```
CHGAUT OBJ('/QSYS.LIB/WWW.LIB') USER(QTMHHTP1) DTAAUT(*RX)
CHGAUT OBJ('/QSYS.LIB/WWW.LIB/MACRO.FILE') USER(QTMHHTP1) DTAAUT(*RX)
```

Punkte, die bei der sprachumgebungsspezifischen Berechtigung zu berücksichtigen sind, werden in den einzelnen Sprachumgebungsabschnitten in „Verwenden der Sprachumgebungen“ auf Seite 91 erörtert.

Sichern der Datenbestände

Internet-Sicherheit wird durch eine Kombination aus Firewall-Technologie, Betriebssystemfunktionen, Web-Server-Funktionen, Net.Data-Mechanismen und Zugriffssteuerungsmechanismen, die Teil Ihrer Datenquellen sind, zur Verfügung gestellt.

Sie müssen entscheiden, welche Sicherheitsstufe für Ihre Datenbestände angebracht ist. In diesem Kapitel werden Methoden zur Sicherung Ihrer Datenbestände beschrieben und Verweise auf zusätzliche Quellen gegeben, mit denen Sie die Sicherheit Ihrer Web-Site planen können.

In den folgenden Abschnitten werden Richtlinien für den Schutz Ihrer Datenbestände erläutert. Folgende Sicherheitsmechanismen werden beschrieben:

- „Verwenden von Firewalls“
- „Verschlüsseln Ihrer Daten im Netzwerk“ auf Seite 29
- „Verwenden der Authentifizierung“ auf Seite 30
- „Verwenden der Berechtigung“ auf Seite 31
- „Verwenden von Net.Data-Mechanismen“ auf Seite 31

Verwenden von Firewalls

Firewalls sind Gruppen von Hardware, Software und Maßnahmen, mit denen der Zugriff auf Ressourcen in einer Netzwerkumgebung eingeschränkt werden soll.

Firewalls haben folgende Funktionen:

- Schützen des internen Netzwerks vor unbefugtem Zugriff oder Störung
- Schützen des internen Netzwerks vor Daten und Programmen, die durch interne Benutzer eingeführt werden
- Begrenzen des Zugriffs interner Benutzer auf externe Daten
- Beschränken des möglichen Schadens bei einem möglichen Durchbrechen der Firewall

Net.Data kann mit Firewall-Produkten verwendet werden, die in Ihrer Umgebung ausgeführt werden.

Die folgenden Konfigurationsmöglichkeiten sind Empfehlungen für die Verwaltung der Sicherheit Ihrer Net.Data-Anwendung. Diese Konfigurationsmöglichkeiten enthalten wertvolle Informationen, wobei davon ausgegangen wird, daß Sie bereits eine Firewall konfiguriert haben, mit der Sie Ihr sicheres Intranet vom öffentlich zugänglichen Internet abgrenzen. Prüfen Sie sorgfältig, welche der folgenden Konfigurationen am besten für die in Ihrem Unternehmen eingesetzten Sicherheitsmaßnahmen geeignet ist.

- **Konfiguration mit hoher Sicherheit**

Mit dieser Konfiguration wird ein Teilnetzwerk erstellt, mit dem Net.Data und der Web-Server sowohl vom sicheren Intranet als auch vom öffentlichen Internet abgegrenzt werden. Die Firewall-Software wird für die Erstellung einer ersten Firewall zwischen Web-Server und öffentlichem Internet sowie einer zweiten Firewall zwischen Web-Server und gesichertem Intranet, in dem sich der DB2-Server befindet, verwendet. Abb. 3 zeigt diese Konfiguration.

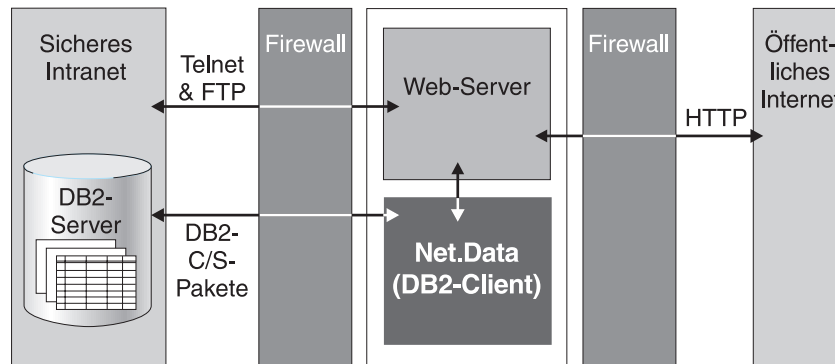


Abbildung 3. Konfiguration mit hoher Sicherheit

Gehen Sie wie folgt vor, um diese Konfiguration zu definieren:

- Installieren Sie Net.Data auf der Web-Server-Maschine, und stellen Sie sicher, daß Net.Data innerhalb des Intranets auf den DB2-Server zugreifen kann. Konfigurieren Sie dafür die Firewall so, daß ein DB2-Datenverkehr durch die Firewall möglich ist. Eine Möglichkeit besteht darin, Regeln zur Paketfilterung hinzuzufügen, die DB2-Client-Anforderungen von Net.Data und Bestätigungspakete vom DB2-Server an Net.Data durchlassen.
- Erlauben Sie FTP- und Telnet-Zugriffe zwischen dem Web-Server und dem sicheren Intranet. Eine Möglichkeit besteht darin, einen Socks-Server auf der Web-Server-Maschine zu installieren.
- Geben Sie in der Konfigurationsdatei für die Paketfilterung der Firewall-Software an, daß eingehende TCP-Pakete vom HTTP-Standardanschluß auf den Web-Server zugreifen dürfen. Geben Sie weiterhin an, daß abgehende TCP-Bestätigungspakete vom Web-Server an jeden beliebigen Host des öffentlichen Internets gesendet werden dürfen.

- **Konfiguration mit mittlerer Sicherheit**

Bei dieser Konfiguration trennt die Firewall-Software das gesicherte Intranet mit dem DB2-Server vom öffentlichen Internet. Net.Data und der Web-Server befinden sich außerhalb der Firewall auf einer Workstation-Plattform. Diese Konfiguration ist einfacher als die oben beschriebene, bietet jedoch trotzdem einen Datenbankschutz. Abb. 4 auf Seite 29 zeigt diese Konfiguration.

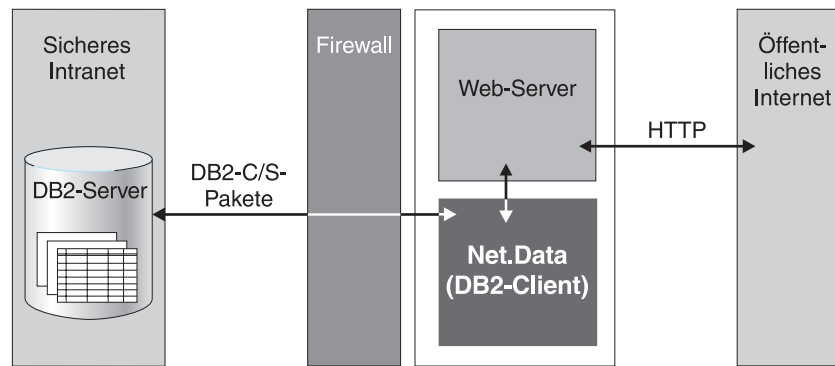


Abbildung 4. Konfiguration mit mittlerer Sicherheit

Die Firewall muß so konfiguriert sein, daß DB2-Client-Anforderungen von Net.Data an DB2 weitergeleitet werden können und Bestätigungspakete von DB2 an Net.Data durchgelassen werden.

- **Konfiguration mit niedriger Sicherheit**

Bei dieser Konfiguration werden der DB2-Server und Net.Data außerhalb der Firewall und des gesicherten Intranets installiert. Dadurch sind sie nicht gegen externe Manipulation geschützt. Für diese Konfiguration benötigt die Firewall keine Regeln zur Paketfilterung. Abb. 5 zeigt diese Konfiguration.

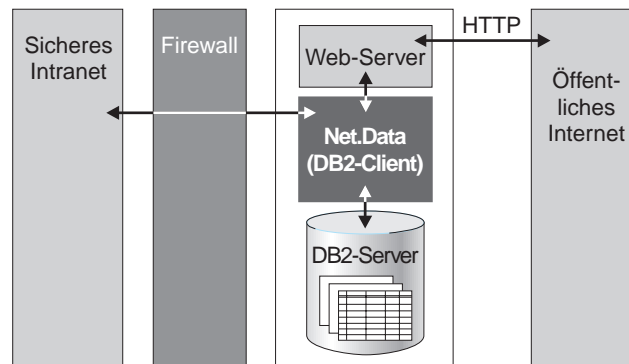


Abbildung 5. Konfiguration mit niedriger Sicherheit

Verschlüsseln Ihrer Daten im Netzwerk

Wenn Sie einen Web-Server verwenden, der SSL (Secured Sockets Layer) unterstützt, können Sie alle Daten, die zwischen einem Client-System und Ihrem Web-Server gesendet werden, verschlüsseln. Diese Sicherheitsmaßnahme unterstützt die Verschlüsselung von Anmelde-IDs, Kennwörtern und aller Daten, die über HTML-Formulare vom Client-System an den Web-Server übertragen werden, sowie aller Daten, die vom Web-Server an das Client-System gesendet werden.

Verwenden der Authentifizierung

Authentifizierung wird verwendet, um sicherzustellen, daß eine Benutzer-ID, die eine Net.Data-Anforderung absetzt, berechtigt ist, auf Daten in der Anwendung zuzugreifen und sie zu aktualisieren. Bei der Authentifizierung wird die Benutzer-ID mit einem Kennwort abgeglichen, um zu überprüfen, ob die Anforderung von einer gültigen Benutzer-ID stammt. Der Web-Server ordnet jeder Net.Data-Anforderung, die er verarbeitet, eine Benutzer-ID zu. Der Prozeß bzw. Thread, der die Anforderung bearbeitet, kann dann auf eine beliebige Ressource zugreifen, für die diese Benutzer-ID über eine entsprechende Berechtigung verfügt.

In einer OS/400-Umgebung kann eine Benutzer-ID dem Thread bzw. Prozeß zugeordnet werden, der eine Net.Data-Anforderung auf eine der drei folgenden Arten handhabt:

Client-abhängige Authentifizierung

Der Benutzer wird auf dem Client aufgefordert, die Benutzer-ID und das Kennwort für ein lokales OS/400-System anzugeben. Der Web-Server führt dann für den Benutzer eine Authentifizierung aus. Bei einer erfolgreichen Authentifizierung wird die angegebene Benutzer-ID der Anforderung zugeordnet. Die Verwendung der speziellen Benutzer-ID für Web-Server-Zugriffssteuerung `%%CLIENT%%` ermöglicht diese Art von Authentifizierung.

Die client-abhängige Authentifizierung wird vom IBM HTTP-Server beginnend mit OS/400 V4R1 unterstützt.

Server-abhängige Authentifizierung

Die Benutzer-ID des Web-Servers wird jeder Anforderung zugeordnet, und der Benutzer wird nicht nach Benutzer-ID oder Kennwort gefragt. Die Verwendung der speziellen Benutzer-ID für Web-Server-Zugriffssteuerung `%%SERVER%%` ermöglicht diese Art von Authentifizierung.

In der Standardeinstellung führt der IBM HTTP-Server GCI-Programme unter der Benutzer-ID (dem Benutzerprofil) QTMHHTTP1 aus. Wenn jedoch die Anweisung `UserID` wirksam ist oder in einer Konfiguration mit Zugriffsschutz, bei der die Unteranweisung `UserID` angegeben wurde, wird das Programm unter der angegebenen Benutzer-ID ausgeführt.

Ersatzauthentifizierung

Eine Ersatzbenutzer-ID mit der Berechtigung zum Zugriff auf einige vordefinierte Ressourcensammlungen wird der Client-Anforderung zugeordnet. Für diese Art von Authentifizierung ist das Erstellen von Ersatzbenutzer-IDs mit einer Zugriffsberechtigung erforderlich, die für eine Benutzergruppe oder Anforderungsklasse geeignet ist. Bei der Authentifizierung mit Ersatzbenutzer-IDs werden gewöhnlich Gültigkeitsprüfungslistenobjekte verwendet, die in Version 4 Release 1 eingeführt wurden. Weitere Informationen und Beispiele finden Sie im Handbuch *OS/400 System API Reference*.

Die Vorgehensweise, die der Web-Server für das Zuordnen einer Benutzer-ID zu einer Client-Anforderung verwendet, wird während der Konfiguration des Web-Servers angegeben. Weitere Informationen zu Benutzer-IDs für Zugriffssteuerung, zum Installieren des Web-Servers und zur Verwendung der Anweisungen „Protect“, „Protection“, „DefProt“ und „UserId“ für die Konfiguration des Web-Servers finden Sie in der Dokumentation zu Ihrem HTTP-Server.

Hinweis .Gehen Sie wie folgt vor, um Net.Data-Makros zu schützen:

1. Fügen Sie Zugriffsschutzanweisungen für das Net.Data-Programmdateiobjekt in die Web-Server-Konfigurationsdatei ein.
2. Stellen Sie sicher, daß die Benutzer-ID, unter der Net.Data ausgeführt wird, Zugriffsrechte für die Makrodateien besitzt. Weitere Informationen zur Erteilung von Zugriffsrechten finden Sie in „Erteilen von Zugriffsrechten für Objekte, auf die Net.Data zugreift“ auf Seite 25.

Verwenden der Berechtigung

Eine *Berechtigung* erlaubt einem Benutzer den vollständigen oder beschränkten Zugriff auf ein Objekt, eine Ressource oder Funktion. Datenquellen wie DB2 stellen ihre eigenen Berechtigungsmechanismen zum Schutz der von Ihnen verwalteten Daten zur Verfügung. Diese Mechanismen gehen davon aus, daß für die Benutzer-ID des Prozesses, der die Net.Data-Anforderung ausführt, eine ordnungsgemäße Authentifizierung ausgeführt wurde.

Eine genauere Erklärung hierzu finden Sie in „Verwenden der Authentifizierung“ auf Seite 30. Die vorhandenen Zugriffssteuerungsmechanismen für diese Datenquellen erteilen bzw. verweigern dann je nach den Berechtigungen der überprüften Benutzer-ID den Zugriff.

Verwenden von Net.Data-Mechanismen

Zusätzlich zu den oben beschriebenen Methoden können Sie mit Net.Data-Konfigurationsvariablen oder Makro-Entwicklungsverfahren die Aktivitäten von Endbenutzern begrenzen, um firmeninterne Informationen wie den Aufbau Ihrer Datenbank zu verdecken und um vom Benutzer gestellte Eingabewerte in Produktionsumgebungen zu überprüfen.

Net.Data-Konfigurationsvariablen

Net.Data stellt mehrere Konfigurationsvariablen zur Verfügung, mit denen Sie die Aktivitäten von Endbenutzern begrenzen oder den Aufbau Ihrer Datenbank verdecken können.

Steuern des Dateizugriffs mit Pfadanweisungen

Net.Data überprüft die Einstellungen der Pfadkonfigurationsanweisungen, um die Speicherposition der Dateien und ausführbaren Programme zu ermitteln, die von Net.Data-Makros verwendet werden. Diese Pfadanweisungen geben eines oder mehrere Verzeichnisse an, die Net.Data durchsucht, wenn es versucht, Makrodateien, ausführbare Dateien, Kopfdateien oder andere unstrukturierte Dateien zu lokalisieren. Durch die selektive Aufnahme von Verzeichnissen in diese Pfadanweisungen können Sie explizit steuern, auf welche Dateien die Benutzer über Browser zugreifen können. Zusätzliche Informationen zu Pfadanweisungen finden Sie in „Konfigurieren von Net.Data“ auf Seite 5.

Sie sollten zudem Berechtigungsprüfungen verwenden (siehe „Verwenden der Berechtigung“) und sicherstellen, daß Dateinamen in INCLUDE-Anweisungen nicht geändert werden können (siehe „Makro-Entwicklungsverfahren“ auf Seite 32).

Inaktivieren von SHOWSQL für Produktionssysteme

Mit der Variable SHOWSQL kann der Benutzer angeben, daß Net.Data die in Net.Data-Funktionen angegebenen SQL-Anweisungen in einem Web-Browser anzeigt. Diese Variable wird hauptsächlich zum Entwickeln und Testen von SQL in einer Anwendung verwendet und wurde nicht zur Verwendung in Produktionssystemen konzipiert.

Sie können die Anzeige von SQL-Anweisungen in Produktionsumgebungen mit einer der folgenden Methoden inaktivieren:

- Bei der Verwendung von Net.Data-Versionen, die die Konfigurationsvariable DTW_SHOWSQL unterstützen, können Sie mit dieser Variable in der Net.Data-Initialisierungsdatei die Auswirkung der Einstellung SHOWSQL in Ihren Net.Data-Makros überschreiben.

Informationen zur Syntax und andere nützliche Informationen finden Sie in „DTW_SHOWSQL: Konfigurationsvariable zum Aktivieren bzw. Inaktivieren von SHOWSQL“ auf Seite 10.

- Verwenden Sie die Funktion DTW_ASSIGN() (siehe „Makro-Entwicklungsverfahren“).

Informationen zur Syntax und Beispiele für die Net.Data-Variable SHOWSQL finden Sie unter SHOWSQL im Kapitel über Variablen des Handbuchs *Net.Data Reference*.

Makro-Entwicklungsverfahren

Net.Data stellt mehrere Mechanismen bereit, mit denen Benutzer Eingabevariablen Werte zuordnen können. Sie können sicherstellen, daß Makros auf die gewünschte Art ausgeführt werden, indem Sie diese Eingabevariablen durch das Makro prüfen lassen. Ihre Datenbank und Anwendung sollten zudem so entworfen werden, daß der Zugriff von Benutzern auf Daten, zu deren Anzeige sie berechtigt sind, begrenzt ist.

Verwenden Sie beim Schreiben Ihrer Net.Data-Makros folgende Entwicklungsverfahren. Diese Verfahren tragen dazu bei, daß Ihre Anwendungen wie gewünscht ausgeführt werden und daß der Datenzugriff auf Benutzer mit entsprechender Berechtigung begrenzt ist.

Sicherstellen, daß Net.Data-Variablen in einer URL-Adresse nicht überschrieben werden können

Die Einstellung von Net.Data-Variablen durch einen Benutzer in einer URL-Adresse überschreibt die Auswirkung von DEFINE-Anweisungen, mit denen Variablen in einem Makro initialisiert werden. Dadurch wird eventuell die Art der Makroausführung geändert. Sie können dies verhindern, indem Sie Ihre Net.Data-Variablen mit der Funktion DTW_ASSIGN() initialisieren.

Beispiel: Verwenden Sie @DTW_ASSIGN(SHOWSQL, "NO") anstelle von %DEFINE SHOWSQL="NO", um die Net.Data-Variable SHOWSQL festzulegen. In diesem Fall überschreibt eine Abfragezeichenfolgezuordnung wie SHOWSQL=YES die Makroeinstellung nicht.

Sie können die Anzeige von SQL-Anweisungen in Produktionsumgebungen mit einer der folgenden Methoden inaktivieren:

- Bei der Verwendung von Net.Data-Versionen, die die Konfigurationsvariable DTW_SHOWSQL unterstützen, können Sie mit dieser Variable in der Net.Data-Initialisierungsdatei die Auswirkung der Einstellung SHOWSQL in Ihren Net.Data-Makros überschreiben. Informationen zur Syntax und andere nützliche Informationen finden Sie in „DTW_SHOWSQL: Konfigurationsvariable zum Aktivieren bzw. Inaktivieren von SHOWSQL“ auf Seite 10.
- Ordnen Sie den Wert für SHOWSQL mit der im obigen Beispiel beschriebenen Funktion DTW_ASSIGN() zu, um eine Überschreibung zu vermeiden.

Informationen zur Syntax und Beispiele für die Net.Data-Variable SHOWSQL finden Sie unter SHOWSQL im Kapitel über Variablen des Handbuchs *Net.Data Reference*.

Mit DTW_ASSIGN können Sie auch sicherstellen, daß andere Net.Data-Variablen wie RPT_MAX_ROWS oder START_ROW_NUM nicht überschrieben werden. Weitere Informationen zu diesen Variablen finden Sie im entsprechenden Kapitel über Variablen des Handbuchs *Net.Data Reference*.

Sicherstellen, daß Ihre SQL-Anweisungen nicht so geändert werden können, daß sich das beabsichtigte Verhalten Ihrer Anwendung ändert

Durch das Hinzufügen einer Net.Data-Variablen zu einer SQL-Anweisung in einem Makro können Benutzer die SQL-Anweisung vor ihrer Ausführung dynamisch ändern. Der Makroautor ist dafür verantwortlich, die vom Benutzer gestellten Eingabewerte zu prüfen und sicherzustellen, daß eine SQL-Anweisung mit einem Variablenverweis nicht auf unerwartete Weise geändert wird. Ihre Net.Data-Anwendung sollte vom Benutzer gestellte Eingabewerte in der URL-Adresse prüfen, so daß die Net.Data-Anwendung ungültige Eingaben zurückweisen kann. Beim Entwerfen einer Gültigkeitsprüfung sollten Sie die folgenden Schritte ausführen:

1. Geben Sie die Syntax für gültige Eingaben an. Zum Beispiel muß eine Kunden-ID mit einem Buchstaben anfangen und kann nur alphanumerische Zeichen enthalten.
2. Ermitteln Sie, welcher Schaden durch versehentliche oder absichtliche falsche Eingaben sowie durch Eingaben, durch die auf interne Daten der Net.Data-Anwendung zugegriffen werden soll, verursacht werden kann.
3. Nehmen Sie in das Makro Eingabeprüfungsanweisungen auf, die die Anforderungen der Anwendung erfüllen. Eine derartige Prüfung hängt von der Syntax der Eingabe und ihrer Verwendungsweise ab. In einfacheren Fällen reicht es eventuell aus, die Eingabe auf ungültigen Inhalt zu überprüfen oder Net.Data aufzurufen, um den Typ der Eingabe zu prüfen. Wenn die Syntax der Eingabe komplexer gestaltet ist, muß der Makroentwickler eventuell die Eingabe teilweise oder vollständig syntaktisch analysieren, um zu prüfen, ob sie gültig ist.

Beispiel 1: Verwenden der Zeichenfolgefunktion DTW_POS() zum Prüfen von SQL-Anweisungen

```
%FUNCTION(DTW_SQL) query1() {  
    select * from shopper where shlogid = '${shlogid}'  
}%
```

Der Wert der Variable shlogid soll eine Käufer-ID (shopper) sein. Durch diese Variable soll die Anzahl der Zeilen, die durch die Anweisung SELECT zurückgegeben wird, auf die Zeilen begrenzt werden, die Informationen zum durch die Käufer-ID angegebenen Käufer enthalten. Wenn jedoch die Zeichenfolge „smith' or shlogid<>'smith“ als Wert der Variable shlogid übergeben wird, sieht die Abfrage wie folgt aus:

```
select * from shopper where shlogid = 'smith' or shlogid<>'smith'
```

Diese vom Benutzer geänderte Version der ursprünglichen SQL-Anweisung SELECT gibt die gesamte Käufertabelle zurück.

Mit den Net.Data-Zeichenfolgefunktionen können Sie sicherstellen, daß die SQL-Anweisung nicht auf unerwünschte Weise durch den Benutzer geändert wird. Sie können z. B. mit der folgenden Logik sicherstellen, daß der Eingabewert, der der Variablen shlogid zugeordnet ist, aus einer einzelnen Käufer-ID besteht:

```
@DTW_POS(" ", $(shlogid), result)  
%IF (result == "0")  
    @query1()  
%ELSE  
    %{ perform some sort of error processing %}  
%ENDIF
```

Beispiel 2: Verwenden von DTW_TRANSLATE()

Angenommen, Ihre Anwendung muß sicherstellen, daß der in der Eingabevariablen number_of_orders bereitgestellte Wert eine ganze Zahl ist. Hierzu können Sie die Umsetztabelle input_translation_table erstellen, die alle Tastaturzeichen mit Ausnahme der numerischen Zeichen 0-9 enthält, und die Eingabe mit den Zeichenfolgefunktionen DTW_TRANSLATE und DTW_POS prüfen:

```
@DTW_TRANSLATE(number_of_orders, "x",  
input_translation_table, "x", string_out)  
@DTW_POS("x", string_out, result)  
    %IF (result = "0")  
        %{ continue with normal processing %}  
%ELSE  
    %{ perform some sort of error processing %}  
%ENDIF
```

Beachten Sie, daß SQL-Anweisungen in gespeicherten Prozeduren nicht durch Benutzer über Web-Browser geändert werden können und daß vom Benutzer gestellte Eingabeparameterwerte durch die SQL-Datentypen, die den Eingabeparametern zugeordnet sind, beschränkt sind. In Situationen, in denen es unpraktisch ist, Benutzereingabewerte mit den Net.Data-Zeichenfolgefunktionen zu prüfen, können Sie gespeicherte Prozeduren verwenden.

Sicherstellen, daß ein Dateiname in einer INCLUDE-Anweisung nicht so geändert wird, daß sich das beabsichtigte Verhalten Ihrer Anwendung ändert

Wenn Sie den Wert für den Dateinamen mit einer INCLUDE-Anweisung bei Verwendung einer Net.Data-Variablen angeben, dann wird die aufzunehmende Datei erst während der Ausführung der INCLUDE-Datei ermittelt. Soll der Wert dieser Variablen in Ihrem Makro festgelegt werden, ein Benutzer jedoch nicht in der Lage sein, den vom Makro bereitgestellten Wert über einen Browser zu überschreiben, dann legen Sie den Wert der Variablen mit DTW_ASSIGN anstelle von DEFINE fest. Wenn der Benutzer in der Lage sein soll, über einen Browser einen Wert für den Dateinamen bereitzustellen, dann sollte Ihr Makro den angegebenen Wert prüfen.

Beispiel: Eine Abfragezeichenfolgezuordnung wie filename=".././x" kann zur Aufnahme einer Datei aus einem Verzeichnis führen, das normalerweise nicht in der Konfigurationsanweisung INCLUDE_PATH angegeben ist. Angenommen, Ihre Net.Data-Initialisierungsdatei enthält die folgende Pfadkonfigurationsanweisung:

```
INCLUDE_PATH /usr/lpp/netdata/include
```

Und angenommen, Ihr Net.Data-Makro enthält die folgende INCLUDE-Anweisung:

```
%INCLUDE "${filename}"
```

Die Abfragezeichenfolgezuordnung filename=".././x" würde die Datei /usr/lpp/x enthalten, was durch die Angabe der Konfigurationsanweisung INCLUDE_PATH nicht beabsichtigt war.

Mit den Net.Data-Zeichenfolgefunktionen können Sie prüfen, ob der bereitgestellte Dateiname für die Anwendung geeignet ist. Sie können z. B. mit der folgenden Logik sicherstellen, daß der Eingabewert, der der Dateinamenvariablen zugeordnet ist, nicht die Zeichenfolge ".." enthält:

```
@DTW_POS("..", ${filename}), result)
%IF (result > "0")
    %{ perform some sort of error processing %}
%ELSE
    %{ continue with normal processing %}
%ENDIF
```

Entwerfen Ihrer Datenbank und Abfragen, so daß Benutzeranforderungen keinen Zugriff auf sensible Daten anderer Benutzer haben

Einige Datenbankentwürfe sammeln sensible Benutzerdaten in einer einzigen Tabelle. Sofern SQL-Anforderungen SELECT nicht auf eine gewisse Art qualifiziert sind, hat diese Vorgehensweise zur Folge, daß ein beliebiger Benutzer über einen Web-Browser eventuell Zugriff auf alle sensiblen Daten hat.

Beispiel: Die folgende SQL-Anweisung gibt Auftragsinformationen zu einem durch die Variable `order_rn` angegebenen Auftrag zurück:

```
select setsstatcode, setsfailtype, mestname
      from merchant, setstatus
     where merfnbr   = setsmenbr
        and setsornbr = $(order_rn)
```

Diese Methode ermöglicht Benutzern, über einen Browser willkürliche Auftragsnummern anzugeben und möglicherweise sensible Informationen zu den Aufträgen anderer Kunden abzurufen. Sie können sich hiervor zum Beispiel schützen, indem Sie die folgenden Änderungen vornehmen:

- Fügen Sie der Auftragsinformationstabelle eine Spalte hinzu, die den Kunden angibt, der den Auftragsinformationen in einer bestimmten Zeile zugeordnet ist.
- Ändern Sie die SQL-Anweisung `SELECT`, um sicherzustellen, daß `SELECT` durch eine authentifizierte, vom Benutzer über den Browser bereitgestellte Kunden-ID qualifiziert ist.

Wenn z. B. `shlogid` die Spalte mit der Kunden-ID ist, die dem Auftrag zugeordnet ist, und wenn `SESSION_ID` eine `Net.Data`-Variable ist, welche die authentifizierte ID des Benutzers am Browser enthält, dann können Sie die vorherige `SELECT`-Anweisung durch die folgende Anweisung ersetzen:

```
select setsstatcode, setsfailtype, mestname
      from merchant, setstatus
     where merfnbr   = setsmenbr
        and setsornbr = $(order_rn)
        and shlogid  = $(SESSION_ID)
```

Verwenden verdeckter `Net.Data`-Variablen

Mit verdeckten `Net.Data`-Variablen können Sie verschiedene Kenndaten Ihrer `Net.Data`-Makros vor Benutzern schützen, die Ihre HTML-Quelle mit ihrem Web-Browser anzeigen möchten. Sie können zum Beispiel die interne Struktur Ihrer Datenbank verdecken. Weitere Informationen zu verdeckten Variablen finden Sie in „Verdeckte Variablen“ auf Seite 60.

Anfordern von Informationen zur Gültigkeitsprüfung von einem Benutzer

Sie können Ihr eigenes Schutzschema basierend auf der vom Benutzer gestellten Eingabe erstellen. Beispielsweise könnten Sie anhand eines HTML-Formulars Informationen zur Gültigkeitsprüfung von einem Benutzer anfordern und diese Informationen dann anhand von Daten, die Ihr `Net.Data`-Makro aus einer Datenbank abrufen, oder durch Aufrufen eines externen Programms aus einer Funktion, die in Ihrem `Net.Data`-Makro definiert ist, überprüfen lassen.

Weitere Informationen zum Schutz Ihrer Datenbestände enthält die Internet-Liste zu häufig gestellten Sicherheitsfragen, die Sie unter folgender Web-Adresse finden:

<http://www.w3.org/Security/Faq>

Aufrufen von Net.Data

Net.Data für OS/400 wird mit CGI (Common Gateway Interface) und einem Makro aufgerufen. Diese Art der Aufrufmethode wird Makroanforderung genannt. Außerdem können Sie permanente Makros oder Makros aufrufen, die Funktionen enthalten, die Transaktionsgrenzen festlegen. Weitere Informationen zu permanenten Makros finden Sie in „Transaktionsverwaltung mit permanenten Makros“ auf Seite 123.

In diesem Kapitel wird das Aufrufen von Net.Data mit einem Makro beschrieben.

- „Aufrufen von Net.Data mit einem Makro (Makroanforderung)“
- „Aufrufen eines permanenten Makros“ auf Seite 40

Aufrufen von Net.Data mit einem Makro (Makroanforderung)

In diesem Abschnitt wird gezeigt, wie Sie Net.Data durch Angabe eines Makros aufrufen können.

Die folgenden Syntaxanweisungen zeigen, wie Net.Data aufgerufen werden kann.

- URL-Adresse:

`http://server/Net.Data_invocation_path/filename/block[?name=val&...]`

Parameter:

server Gibt den Namen und Pfad des Web-Servers an. Wenn es sich bei dem Server um den lokalen Server handelt, kann der Server-Name übergangen und eine relative URL-Adresse verwendet werden.

Net.Data_invocation_path

Gibt Pfad und Dateinamen der ausführbaren Net.Data-Datei an. Zum Beispiel `/cgi-bin/db2www/`.

filename

Gibt den Namen der Net.Data-Makrodatei an. Net.Data sucht diesen Dateinamen und versucht, ihn mit den Pfadanweisungen abzugleichen, die in der Initialisierungspfadvariablen `MACRO_PATH` definiert sind. Weitere Informationen hierzu finden Sie in „`MACRO_PATH`“ auf Seite 16.

block Gibt den Namen des HTML-Blocks im angegebenen Net.Data-Makro an.

method

Gibt die für das Formular verwendete HTML-Methode an.

?name=val&...

Gibt einen oder mehrere wahlfreie Parameter an, die an Net.Data weitergegeben werden.

Sie können die URL-Adresse direkt in Ihrem Browser angeben, oder Sie können sie wie folgt in einer HTML-Programmverbindung (Link) bzw. in einem HTML-Formular verwenden:

- HTML-Programmverbindung (Link):

`any text`

- HTML-Formular:

`<FORM METHOD=method ACTION="URL">any text</FORM>`

Parameter:

<i>method</i>	Gibt die für das Formular verwendete HTML-Methode an.
<i>URL</i>	Gibt die URL-Adresse an, mit der das Net.Data-Makro ausgeführt wird und deren Parameter oben beschrieben sind.

Beispiele

Die folgenden Beispiele veranschaulichen die verschiedenen Methoden zum Aufrufen von Net.Data.

Beispiel 1: Aufrufen von Net.Data mit einer HTML-Programmverbindung (Link)

```
<A HREF="http://server/cgi-bin/db2www/myMacro.d2w/report">  
.  
.  
.  
</A>
```

Beispiel 2: Aufrufen von Net.Data mit einem Formular

```
<FORM METHOD=POST  
ACTION="http://server/cgi-bin/db2www/myMacro.d2w/report">  
.  
.  
.  
</FORM>
```

Beispiel 3: Aufrufen von Net.Data-Makros im Dateisystem qsys.lib mit einer HTML-Programmverbindung (Link)

```
<A HREF="http://server/cgi-bin/db2www/myMacro.mbr/report">  
.  
.  
.  
</A>
```

Beispiel 4: Aufrufen von Net.Data-Makros im Dateisystem qsys.lib mit einem Formular

```
<FORM METHOD=POST  
ACTION="http://server/cgi-bin/db2www/  
qsys.lib/mylib.lib/myfile.file/myMacro.mbr/report">  
.  
.  
.  
</FORM>
```

In den folgenden Abschnitten werden HTML-Programmverbindungen (Links) und -Formulare und der Aufruf von Net.Data mit diesen Mitteln beschrieben:

- „HTML-Programmverbindungen (Links)“ auf Seite 39
- „HTML-Formulare“ auf Seite 39

HTML-Programmverbindungen (Links)

Wenn Sie eine Web-Seite verfassen, können Sie eine HTML-Programmverbindung (Link) erstellen, die zur Ausführung eines HTML-Blocks führt. Wenn ein Benutzer über einen Browser den Text bzw. das Bild anklickt, der bzw. das als eine HTML-Programmverbindung (Link) definiert ist, führt Net.Data den HTML-Block im Makro aus.

Verwenden Sie den HTML-Befehl `<a>`, um eine HTML-Programmverbindung (Link) zu erstellen. Entscheiden Sie, welcher Text bzw. welche Grafik als Hyperlink zum Net.Data-Makro verwendet werden soll. Stellen Sie ihm bzw. ihr dann den Befehl `<a>` vor und den Befehl `` nach. Geben Sie im Attribut HREF des Befehls `<a>` das Makro und den HTML-Block an.

Im folgenden Beispiel wird eine Programmverbindung (Link) gezeigt, die zur Ausführung einer SQL-Abfrage führt, wenn ein Benutzer den Text "List all monitors" auf einer Web-Seite auswählt.

```
<a href="http://server/cgi-bin/db2www/listA.d2w/report">
List all monitors</a>
```

Durch das Anklicken der Programmverbindung (Link) wird das Makro listA.d2w mit dem HTML-Block "report" aufgerufen. Siehe folgendes Beispiel:

```
%DEFINE DATABASE="MNS97"
```

```
%FUNCTION(DTW_SQL) myQuery(){
SELECT MODNO, COST, DESCRIP FROM EQPTABLE
WHERE TYPE='MONITOR'
%}
```

```
%HTML(report){
@myQuery()
%}
```

Diese Abfrage gibt eine Tabelle mit der Modellnummer (MODNO), dem Preis (COST) und einer Beschreibung (DESCRIP) für jeden Monitor zurück, der in der Tabelle EQPTABLE beschrieben ist. Dieses Beispiel zeigt die Ergebnisse der Abfrage durch Generierung eines Standardberichts an. Informationen zum Anpassen Ihrer Berichte mit einem REPORT-Block finden Sie in „REPORT-Blöcke“ auf Seite 79.

HTML-Formulare

Sie können die Ausführung Ihrer Net.Data-Makros mit HTML-Formularen dynamisch anpassen. Formulare ermöglichen Benutzern die Angabe von Eingabewerten, die die Ausführung des Makros und den Inhalt der von Net.Data erstellten Web-Seite beeinflussen.

Das folgende Beispiel baut auf dem Beispiel zur Monitorliste aus „HTML-Programmverbindungen (Links)“ auf, indem es Benutzern ermöglicht, in einem Browser mit Hilfe eines einfachen HTML-Formulars den Produkttyp auszuwählen, für den Informationen angezeigt werden sollen.

```

<H1>Hardware Query Form</H1>
<HR>
<FORM METHOD=POST ACTION="/cgi-bin/db2www/equip1st.d2w/report">
<P>What type of hardware do you want to see?
<MENU>
<LI><INPUT TYPE="RADIO" NAME="hdware" VALUE="MON" checked> Monitors
<LI><INPUT TYPE="RADIO" NAME="hdware" VALUE="PNT"> Pointing devices
<LI><INPUT TYPE="RADIO" NAME="hdware" VALUE="PRT"> Printers
<LI><INPUT TYPE="RADIO" NAME="hdware" VALUE="SCN"> Scanners
</MENU>

<INPUT TYPE="SUBMIT" VALUE="Submit">
</FORM>

```

Nachdem der Benutzer im Browser seine Auswahl getroffen und den Knopf für die Übergabe angeklickt hat, verarbeitet der Web-Server den Parameter ACTION des Befehls FORM, wodurch Net.Data aufgerufen wird. Net.Data führt anschließend den HTML-REPORT-Block im Makro equip1st.d2w aus:

```

%DEFINE DATABASE="MNS97"

%FUNCTION(DTW_SQL) myQuery(){
SELECT MODNO, COST, DESCRIP FROM EQPTABLE
WHERE TYPE='$ (hdware) '
  %REPORT{
<H3>Here is the list you requested</H3>
  %ROW{
<HR>
$(N1): $(V1), $(N2): $(V2)
<P>$(N3): $(V3)
%}
%}
%}

%HTML(report){
@myQuery()
%}

```

Im obigen Beispiel wird der Wert für TYPE=\$(hdware) der SQL-Anweisung der HTML-Formulareingabe entnommen.

Eine detaillierte Beschreibung der im ROW-Block verwendeten Variablen finden Sie im Handbuch *Net.Data Reference*.

Aufrufen eines permanenten Makros

In diesem Abschnitt wird gezeigt, wie Sie permanente Makros aufrufen können. Diese Makros enthalten Funktionen, die für die Transaktionsverarbeitung verwendet werden. Der Aufruf dieser Makros ist ähnlich wie eine normale Makroanforderung, bei der Sie einen Server, ein Makro und einen HTML-Block angeben. Bei permanenten Makros geben Sie auch eine Transaktionskennung an, die den HTML-Block als Bestandteil einer Transaktion angibt.

Weitere Informationen zu permanenten Makros und der Transaktionsverarbeitung finden Sie in „Transaktionsverwaltung mit permanenten Makros“ auf Seite 123.

Syntax permanenter Makros

Verwenden Sie die folgende Syntax zum Aufrufen eines permanenten Makros:

- HTML-Programmverbindung (Link):

```
<A HREF="http://server/Net.Data_invocation_path  
/transaction_handle/filename/  
block/[?name=val&...]">any text</A>
```

- HTML-Formular:

```
<FORM METHOD=method ACTION="http://server/Net.Data_invocation_path/  
transaction_handle/filename/block/  
[?name=val&...]">any text</FORM>
```

- URL-Adresse:

```
http://server/Net.Data_invocation_path/transaction_handle  
/filename/block/  
[?name=val&...]
```

Parameter:

server Gibt den Namen des Web-Servers an. Wenn es sich bei dem Server um den lokalen Server handelt, kann der Server-Name übergangen und eine relative URL-Adresse verwendet werden.

Net.Data_invocation_path

Gibt Pfad und Dateinamen der ausführbaren Net.Data-Datei an. Zum Beispiel /cgi-bin/db2www/.

transaction_handle

Gibt an, welche URL-Adressen Bestandteil einer Transaktion sind, die von einem Net.Data-Makro eingeleitet wird. Diese Kennung wird durch Aufruf der integrierten Funktion DTW_RTVHANDLE ermittelt und muß nach *Net.Data_invocation_path* stehen.

filename

Gibt den Namen der Net.Data-Makrodatei an. Net.Data sucht diesen Dateinamen und versucht, ihn mit den Pfadanweisungen abzugleichen, die in der Initialisierungspfadvariablen MACRO_PATH definiert sind. Weitere Informationen hierzu finden Sie in „MACRO_PATH“ auf Seite 16.

block Gibt den Namen des HTML-Blocks im angegebenen Net.Data-Makro an.

method

Gibt die für das Formular verwendete HTML-Methode an.

?name=val&...

Gibt einen oder mehrere wahlfreie Parameter an, die an Net.Data weitergegeben werden.

Beispiele

Die folgenden Beispiele zeigen, wie Sie permanente Makros aufrufen können.

Beispiel 1: Eine URL-Adresse in einem Makro

```
http://www.mycompany.com/cgi-bin/db2www/$(handle)/mymacro.mac/report1
```

Beispiel 2: Ein typischer HTML-Block mit Programmverbindungen (Links) zu anderen Makroaufrufen, die in der gleichen Transaktion ausgeführt werden

```
@DTW_STATIC()
...
%define handle = ""
@DTW_RTVHANDLE(handle)

%html (report) {
@DTW_ACCEPT(handle)
...
<a href="/cgi-bin/db2www/$(handle)/qsys.lib/mylib.lib/macros.file/
  pcgil.mbr/report2">continue</a><br>
<a href="/cgi-bin/db2www/$(handle)/qsys.lib/mylib.lib/macros.file/
  pcgil.mbr/quit">quit</a><br>
%}
```

Entwickeln von Net.Data-Makros

Ein Net.Data-Makro ist eine Textdatei, die aus einer Reihe von Net.Data-Makrosprachkonstrukten besteht, die folgenden Zwecken dienen:

- Angeben des Layouts von Web-Seiten
- Definieren von Variablen und Funktionen
- Aufrufen von Funktionen, die in Net.Data integriert bzw. im Makro definiert sind
- Formatieren der Verarbeitungsausgabe in HTML und Rückgabe an den Web-Browser zur Anzeige

Das Net.Data-Makro enthält zwei Abschnitte: den Deklarationsabschnitt und den Darstellungsabschnitt, wie in Abb. 6 gezeigt wird.

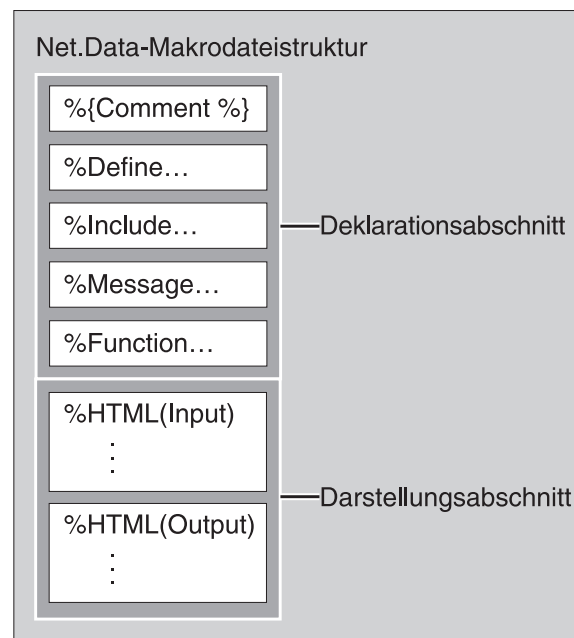


Abbildung 6. Makrostruktur

- Der *Deklarationsabschnitt* enthält die Definitionen von Variablen und Funktionen im Makro.
- Der *Darstellungsabschnitt* enthält HTML-Blöcke, die das Layout der Web-Seite festlegen. Die HTML-Blöcke bestehen aus Anweisungen zur Textdarstellung, die von Ihrem Web-Browser unterstützt werden, etwa HTML und JavaScript.

Sie können diese Abschnitte auch mehrmals in beliebiger Reihenfolge verwenden. Im Handbuch *Net.Data Reference* finden Sie Informationen zur Syntax der Makroabschnitte und der Konstrukte.

Hinweis zu Berechtigungen: Stellen Sie sicher, daß der Web-Server Zugriffsrechte für diese Datei hat. Weitere Informationen hierzu finden Sie in „Erteilen von Zugriffsrechten für Objekte, auf die Net.Data zugreift“ auf Seite 25.

Dieses Kapitel behandelt die verschiedenen Blöcke, aus denen ein Net.Data-Makro besteht, und erläutert die Methoden, mit denen Sie das Makro schreiben können.

- „Aufbau eines Net.Data-Makros“
- „Net.Data-Makrovariablen“ auf Seite 51
- „Net.Data-Funktionen“ auf Seite 66
- „Generieren von Web-Seiten in einem Makro“ auf Seite 77
- „Bedingte Logik und Schleifen in einem Makro“ auf Seite 86

Aufbau eines Net.Data-Makros

Das Makro besteht aus zwei Abschnitten:

- Dem Deklarationsabschnitt, der die im Darstellungsabschnitt verwendeten Definitionen enthält. Im Deklarationsabschnitt werden zwei wahlfreie Hauptblöcke verwendet:
 - DEFINE-Block
 - FUNCTION-Block

Der Deklarationsabschnitt kann darüber hinaus noch weitere Sprachkonstrukte und Anweisungen enthalten, wie z. B. EXEC-Anweisungen, IF-Blöcke, INCLUDE-Anweisungen und MESSAGE-Blöcke. Weitere Informationen zu den Sprachkonstrukten finden Sie im Kapitel über die Sprachkonstrukte im Handbuch *Net.Data Reference*.

Hinweis zu Berechtigungen: Stellen Sie sicher, daß der Web-Server Zugriffsrechte für Dateien hat, auf die in EXEC- und INCLUDE-Anweisungen verwiesen wird. Weitere Informationen hierzu finden Sie in „Erteilen von Zugriffsrechten für Objekte, auf die Net.Data zugreift“ auf Seite 25.

- Der Darstellungsabschnitt definiert das Layout der Web-Seite, verweist auf Variablen und ruft Funktionen mit Hilfe von HTML-Blöcken auf, die als Eingangs- und Endpunkte für das Makro verwendet werden. Wenn Sie Net.Data aufrufen, geben Sie den Namen eines HTML-Blocks als Eingangspunkt zur Verarbeitung des Makros an. Die HTML-Blöcke werden in „HTML-Blöcke“ auf Seite 48 beschrieben.

Im folgenden Abschnitt werden anhand eines einfachen Net.Data-Makros die Elemente der Makrosprache erläutert. Dieses Beispielmakro zeigt ein Formular, das Informationen anfordert, die an ein REXX-Programm übergeben werden. Das Makro übergibt diese Informationen an ein externes REXX-Programm namens `ompsamp .mbr`, das die vom Benutzer eingegebenen Daten zurückmeldet. Die Ergebnisse werden anschließend auf einer zweiten HTML-Seite angezeigt.

Sehen Sie sich zunächst das gesamte Makro an. Im folgenden werden dann die einzelnen Blöcke näher erläutert:

```
%{ ***** DEFINE block *****%}
%DEFINE {
    page_title="Net.Data Macro Template"
%}

%{ ***** FUNCTION Definition block *****%}
%FUNCTION(DTW_REXX) rexx1 (IN input) returns(result)
{
    %EXEC{ompsamp.mbr %}
%}

%FUNCTION(DTW_REXX) today () RETURNS(result)
{
    result = date()
%}

%{ ***** HTML Block: Input *****%}
%HTML(INPUT){
<html>
<head>
<title>$(page_title)</title>
</head><body>
<h1>Input Form</h1>
Today is @today()

<FORM METHOD="post" ACTION="OUTPUT">
Type some data to pass to a REXX program:
<INPUT NAME="input_data" TYPE="text" SIZE="30">
<p>
<INPUT TYPE="submit" VALUE="Enter">

</form>

<hr>
<p>[<a href="/">Home page</a>]
</body></html>
%}

%{ ***** HTML Block: Output *****%}
%HTML (OUTPUT) {
<html>
<head>
<title>$(page_title)</title>
</head><body>
<h1>Output Page</h1>
<p>@rexx1(input_data)
<p><hr>
<p>[<a href="/">Home page</a> |
<a href="input">Previous page</a>]
</body></html>
%}
```

Das Beispielmakro besteht aus vier Hauptblöcken: dem DEFINE-, dem FUNCTION- sowie den beiden HTML-Blöcken. Ein Net.Data-Makro kann auch mehrere DEFINE-, FUNCTION- und HTML-Blöcke enthalten.

Die beiden HTML-Blöcke enthalten Textdarstellungsanweisungen wie HTML, die das Schreiben von Web-Makros sehr einfach machen. Wenn Sie mit HTML vertraut sind, besteht die Erstellung eines Makros nur aus dem Hinzufügen von Makroanweisungen, die dynamisch auf dem Server verarbeitet werden, sowie von SQL-Anweisungen, die an die Datenbank gesendet werden.

Obwohl das Makro einem HTML-Dokument ähnelt, greift der Web-Server über Net.Data mit Hilfe von CGI darauf zu. Net.Data benötigt zum Aufrufen eines Makros zwei Parameter: den Namen des zu verarbeitenden Makros und den anzuzeigenden HTML-Block in diesem Makro.

Wenn das Makro aufgerufen wird, beginnt Net.Data mit der Verarbeitung am Anfang der Datei. In den folgenden Abschnitten wird die Verarbeitung der einzelnen Blöcke durch Net.Data beschrieben.

Der DEFINE-Block

Der DEFINE-Block enthält das DEFINE-Sprachkonstrukt und Variablendefinitionen, die später in den HTML-Blöcken verwendet werden. Das folgende Beispiel zeigt einen DEFINE-Block mit einer Variablendefinition:

```
%{ ***** DEFINE Block *****%}  
%DEFINE {  
    page_title="Net.Data Macro Template"  
%}
```

Die erste Zeile ist ein Kommentar. Ein Kommentar ist jeder Text, der in %{ und %} eingeschlossen ist. Kommentare können an jeder beliebigen Stelle im Makro eingefügt werden. Die nächste Anweisung beginnt einen DEFINE-Block. Sie können mehrere Variablen in einem DEFINE-Block definieren. Im vorliegenden Beispiel wird lediglich eine Variable (page_title) definiert. Nach ihrer Definition kann auf diese Variable an jeder Stelle innerhalb des Makros mit der Syntax \$(page_title) verwiesen werden. Mit Hilfe der Variablen können Sie zu einem späteren Zeitpunkt auf einfache Art globale Änderungen an Ihrem Makro vornehmen. Die letzte Zeile dieses Blocks, %}, kennzeichnet das Ende des DEFINE-Blocks.

Der FUNCTION-Block

Der FUNCTION-Block enthält die Deklarationen für Funktionen, die von den HTML-Blöcken aufgerufen werden. Funktionen werden von Sprachumgebungen verarbeitet und können Programme, SQL-Abfragen oder gespeicherte Prozeduren ausführen.

Das folgende Beispiel zeigt zwei FUNCTION-Blöcke. Der erste Block definiert den Aufruf eines externen REXX-Programms, und der zweite Block enthält Inline-REXX-Anweisungen.

```
%{ ***** FUNCTION Block *****%}
%FUNCTION(DTW_REXX) rexx1 (IN input) returns(result){ <-- Diese Funktion
                                                         akzeptiert einen Parameter
                                                         und gibt die Variable 'result'
                                                         zurück, die vom externen
                                                         REXX-Programm zugeordnet
                                                         wird.

    %EXEC{ompsamp
.mbr %} <-- Die Funktion führt ein externes REXX-Programm
                                                         namens "ompsamp.mbr
                                                         aus.
%}
%FUNCTION(DTW_REXX) today () RETURNS(result){
    result = date() <-- Die einzelne Quellenanweisung für
                                                         diese Funktion befindet sich inline.
%}
```

Der erste FUNCTION-Block, rexx1, ist die Deklaration einer REXX-Funktion, die ihrerseits ein externes REXX-Programm namens ompsamp .mbr ausführt. Von dieser Funktion wird eine Eingabevariable input entgegengenommen und automatisch an den externen REXX-Befehl übergeben. Der REXX-Befehl gibt außerdem eine Variable namens result zurück. Der Inhalt der Variablen result im REXX-

Befehl ersetzt den im OUTPUT-Block enthaltenen Funktionsaufruf @rexx1(). Auf die Variablen input und result kann das REXX-Programm, wie im Quellencode für ompsamp .mbr gezeigt wird, direkt zugreifen:

```
/* REXX */
result = 'The REXX program received "'input'" from the macro.'
```

Der Code in dieser Funktion gibt die an sie übergebenen Daten zurück. Sie können den Ergebnistext nach Belieben formatieren, indem Sie den anfordernden Funktionsaufruf @rexx1() zwischen normale HTML-Befehle für Hervorhebungen (wie z. B. oder) setzen. Anstatt die Variable result zu verwenden, hätte das REXX-Programm auch HTML-Befehle mit Hilfe der REXX-Anweisung SAY in die Standardausgabe schreiben können.

Der zweite FUNCTION-Block, today, verweist ebenfalls auf ein REXX-Programm. In diesem Fall befindet sich jedoch das gesamte REXX-Programm selbst innerhalb der Funktionsdeklaration. Ein externes Programm ist nicht erforderlich. Inline-Programme sind für REXX- und Perl-Funktionen zulässig, da REXX und Perl Interpreter-Sprachen sind, die syntaktisch analysiert und dynamisch ausgeführt werden können. Inline-Programme bieten den Vorteil der Einfachheit, da sie keine separat zu verwaltende Programmdatei erfordern. Die erste REXX-Funktion hätte ebenfalls inline angelegt werden können.

HTML-Blöcke

HTML-Blöcke definieren das Layout einer Web-Seite, verweisen auf Variablen und rufen Funktionen auf. HTML-Blöcke werden als Eingangs- und Endpunkte für das Makro verwendet. In der Net.Data-Aufrufanforderung wird immer ein HTML-Block angegeben, und jedes Makro muß mindestens einen HTML-Block enthalten.

Der erste HTML-Block im Beispielmakro heißt INPUT. HTML(INPUT) enthält HTML-Code für ein einfaches Formular mit einem Eingabefeld.

```
%{ ***** HTML Block: Input *****%}
%HTML (INPUT) {      <--- Gibt den Namen dieses HTML-Blocks an.

<html>
<head>
<title>$(page_title)</title> <--- Beachten Sie die Variablensubstitution.
</head><body>
<h1>Input Form</h1>
Today is @today()          <--- Diese Zeile enthält den Aufruf einer Funktion.

<FORM METHOD="post" ACTION="OUTPUT"> <--- Bei Übergabe dieses Formulars
                                   wird der HTML-Block "OUTPUT"
                                   aufgerufen

Type some data to pass to a REXX program:
<INPUT NAME="input_data"      <--- "input_data" wird bei Übergabe
                                   des Formulars
                                   definiert. Auf diese Variable kann an
                                   anderer Stelle des Makros verwiesen werden.
                                   Sie wird mit der Benutzereingabe
                                   initialisiert.

<p>
<INPUT TYPE="submit" VALUE="Enter">
<hr>
<p>
[
<a href="/">Home page</a>]
</body><html>
%}      <--- Beendet den HTML-Block.
```

Der gesamte HTML-Block wird von der HTML-Blockkennung %HTML (INPUT) {...%} eingeschlossen. INPUT gibt den Namen dieses Blocks an. Der Name kann alphanumerische Zeichen, Unterstreichungszeichen oder Punkte enthalten. Der HTML-Befehl <title> enthält ein Beispiel für eine Variablensubstitution. Der Wert der Variablen page_title wird in den Titel des Formulars eingesetzt. Dieser Block enthält außerdem einen Funktionsaufruf. Der Ausdruck @today() ist ein Aufruf an die Funktion today. Diese Funktion wird im Block FUNCTION definiert, der oben beschrieben ist. Net.Data fügt das Ergebnis der Funktion today, d. h. das aktuelle Datum, in den HTML-Text an der Stelle ein, an der sich der Ausdruck @today() befindet.

Der Parameter ACTION der Anweisung FORM zeigt ein Beispiel für die Navigation zwischen HTML-Blöcken bzw. zwischen Makros. Durch den Verweis auf den Namen eines anderen Blocks in einem Parameter ACTION wird bei der Übergabe des Formulars auf diesen Block zugegriffen. Alle Eingabedaten eines HTML-Formulars werden als implizite Variablen an den neuen Block übergeben. Dies gilt für das einzelne Eingabefeld, das in diesem Formular definiert ist. Bei der Über-

gabe des Formulars werden die in diesem Formular eingegebenen Daten in der Variablen *input_data* an den HTML-Block `HTML(OUTPUT)` übergeben.

Sie können über einen relativen Verweis auf HTML-Blöcke in anderen Makros zugreifen, wenn sich diese Makros auf demselben Web-Server befinden. So greift z. B. der ACTION-Parameter ACTION="../othermacro.d2w/main" auf den HTML-Block main im Makro othermacro.d2w zu. Auch hier werden alle in das Formular eingegebenen Daten in der Variablen *input_data* an dieses Makro übergeben.

Beim Aufrufen von Net.Data wird die Variable als Teil der URL-Adresse weitergegeben. Beispiel:

```
<a href="/cgi-bin/db2www/othermacro.d2w/main?input_data=value">Next macro</a>
```

Sie können auf Formulardaten im Makro zugreifen und sie bearbeiten, indem Sie auf den im Formular angegebenen Variablennamen verweisen.

Der nächste HTML-Block des Beispiels ist der Block HTML(OUTPUT). Er enthält HTML-Befehle und Net.Data-Makroanweisungen, die die Verarbeitung der Ausgabe von der Anforderung HTML(INPUT) definieren.

```
%{ ***** HTML Block: Output *****%}
%HTML (OUTPUT) {
<html>
<head>
<title>$(page_title)</title> <--- Weitere Substitution
  </head><body>
<h1>Output Page</h1>
<p>@rex1(input_data) <--- Diese Zeile enthält einen Aufruf der Funktion
                        rex1, die das Argument "input_data" übergibt.
<p>
<hr>
<p>
[
<a href="/">Home page</a> |
<a href="input">Previous page</a>]
%}
```

Wie der Block HTML(INPUT) besteht auch dieser Block aus Standard-HTML mit Net.Data-Makroanweisungen für die Substitution von Variablen und einem Funktionsaufruf. Auch hier wird in der Titelanweisung der Wert der Variablen *page_title* eingesetzt. Außerdem enthält dieser Block ebenfalls einen Funktionsaufruf. In diesem Fall wird die Funktion *rex1* aufgerufen, und der Inhalt der Variablen *input_data*, der von dem im INPUT-Block definierten Formular empfangen wurde, an die Funktion übergeben. Sie können Variablen in beliebiger Zahl an eine Funktion bzw. von einer Funktion übergeben. Die Funktionsdefinition gibt die Anzahl und die Verwendung der übergebenen Variablen an.

Net.Data-Makrovariablen

Mit Net.Data können Sie Variablen in einem Net.Data-Makro definieren und auf diese Variablen verweisen. Darüber hinaus können Sie diese Variablen vom Makro aus an Sprachumgebungen übergeben und von dort empfangen. Net.Data-Token, wie Variablennamen und -werte, sowie Literalzeichenfolgen können bis zu 256 KB Daten enthalten. Für OS/400 wird die maximale Token-Größe durch das Betriebssystem festgelegt. Einzelne Sprachumgebungen können weitere Einschränkungen für die Größe dieser Werte festlegen.

Bei der Definition von Net.Data-Variablen kann die Art der Variable und bei Bedarf ein vordefinierter Wert festgelegt werden. Diese Variablen können je nach Definition in die folgenden Arten untergliedert werden:

- Explizit mit Hilfe der Anweisung DEFINE im DEFINE-Block definierte Variablen
- Vordefinierte Variablen, die von Net.Data zur Verfügung gestellt werden und auf einen bestimmten Wert gesetzt sind. Dieser Wert kann in der Regel nicht geändert werden.
- Implizit definierte Variablen, von denen es vier Arten gibt:
 - Variablen, die zwar nicht explizit definiert sind, jedoch verfügbar gemacht werden, sobald ihnen zum ersten Mal ein Wert zugeordnet wird
 - Parametervariablen, die zu einer Definition im FUNCTION-Block gehören und auf die nur innerhalb eines FUNCTION-Blocks verwiesen werden kann
 - Variablen, die von Net.Data verfügbar gemacht werden und Formulardaten bzw. Abfragezeichenfolgedaten entsprechen
 - Variablen, die einer Net.Data-Tabelle zugeordnet sind und auf die nur innerhalb eines ROW-Blocks oder eines REPORT-Blocks verwiesen werden kann

In den folgenden Abschnitten wird folgendes beschrieben:

- „Geltungsbereich von Kennungen“
- „Definieren von Variablen“ auf Seite 53
- „Verweisen auf Variablen“ auf Seite 55
- „Variablenarten“ auf Seite 57

Geltungsbereich von Kennungen

Eine Kennung, die eine Variable oder einen Funktionsaufruf darstellt, wird *sichtbar*, d. h. im Makro verwendbar, wenn sie deklariert oder von Net.Data verfügbar gemacht wurde. Der Bereich, in dem eine Kennung sichtbar ist, wird als *Geltungsbereich* der Kennung bezeichnet. Es gibt die folgenden fünf Geltungsbereiche:

- Global

Eine Kennung hat einen globalen Geltungsbereich, wenn auf sie von jeder Stelle des Makros aus verwiesen werden kann. Kennungen mit globalem Geltungsbereich sind:

- Integrierte Net.Data-Funktionen
- Formulardaten
- Abfragezeichenfolgedaten
- Variablen, die innerhalb eines HTML-Blocks verfügbar gemacht werden

- Makro

Eine Kennung hat den Geltungsbereich einer Makrodatei, wenn ihre Deklaration außerhalb aller Blöcke erfolgt. Ein Block beginnt mit einer öffnenden geschweiften Klammer ({) und endet mit einem Prozentzeichen und einer schließenden geschweiften Klammer (%}). (Beachten Sie hierbei, daß DEFINE-Blöcke von dieser Definition ausgeschlossen sind und als separate DEFINE-Anweisungen behandelt werden sollten.) Im Gegensatz zu einer Kennung mit einem globalen Geltungsbereich kann auf eine Kennung mit einem Makrogeltungsbereich nur durch Elemente im Makro verwiesen werden, die auf die Deklaration der Kennung folgen.

- FUNCTION-Block bzw. MACRO_FUNCTION-Block

Eine Kennung hat den Geltungsbereich FUNCTION-Block, wenn folgendes zutrifft:

- Die Kennung wird in der Parameterliste der Funktionsdefinition deklariert.
Wenn eine Kennung mit dem gleichen Namen bereits außerhalb der Funktionsdefinition vorhanden ist, verwendet Net.Data die Kennung aus der Funktionsparameterliste innerhalb des Funktionsblocks.
- Die Kennung wird im Funktionsblock angelegt und vor dem Funktionsaufruf weder deklariert noch initialisiert.

Eine Kennung hat nicht den Geltungsbereich FUNCTION-Block, wenn sie außerhalb der Funktion deklariert oder initialisiert wurde und nicht in der Funktionsparameterliste deklariert wird. Der Wert der Kennung innerhalb des Funktionsblocks bleibt unverändert, außer wenn er durch die Funktion aktualisiert wird.

- REPORT-Block

Eine Kennung hat den Geltungsbereich REPORT-Block, wenn auf sie nur innerhalb eines REPORT-Blocks verwiesen werden kann (zum Beispiel Tabellenspaltennamen N1, N2, ..., Nn). Nur solche Variablen, die Net.Data implizit als Teil der Tabellenverarbeitung definiert, können den Geltungsbereich REPORT-Block haben. Alle anderen verfügbar gemachten Variablen haben den Geltungsbereich FUNCTION-Block.

- ROW-Block

Eine Kennung hat den Geltungsbereich ROW-Block, wenn auf sie nur innerhalb eines ROW-Blocks verwiesen werden kann (zum Beispiel Tabellenwertnamen V1, V2, ..., Vn). Nur solche Variablen, die Net.Data implizit als Teil der Tabellenverarbeitung definiert, können den Geltungsbereich ROW-Block haben. Alle anderen verfügbar gemachten Variablen haben den Geltungsbereich FUNCTION-Block.

Definieren von Variablen

Variablen können in einem Net.Data-Makro auf drei Arten definiert werden:

- DEFINE-Anweisung bzw. -Block
- HTML-Formularbefehle
- Abfragezeichenfolgedaten

Ein Variablenwert, der von einem Formular oder von Abfragezeichenfolgedaten empfangen wird, überschreibt einen mit der Anweisung DEFINE in einem Net.Data-Makro definierten Variablenwert.

- **DEFINE-Anweisung bzw. -Block**

Die einfachste Art, eine Variable zur Verwendung in einem Net.Data-Makro zu definieren, ist der Einsatz der Anweisung DEFINE. Die Syntax sieht wie folgt aus:

```
%DEFINE variable_name="variable value"
```

```
%DEFINE variable_name={ variable value on multiple  
                           lines of text  %}
```

```
%DEFINE {  
    variable_name1="variable value 1"  
    variable_name2="variable value 2"  
%}
```

variable_name ist der Name, den Sie der Variablen geben. Variablennamen müssen mit einem Buchstaben oder Unterstreichungszeichen beginnen und können jedes beliebige alphanumerische Zeichen, ein Unterstreichungszeichen, einen Punkt oder ein Hash-Zeichen (#) enthalten. Alle Variablennamen mit Ausnahme der Tabellenvariablen *N_columnName* und *V_columnName* sind von der Groß-/Kleinschreibung abhängig.

Beispiel:

```
%DEFINE reply="hello"
```

Die Variable *reply* hat den Wert *hello*.

Zwei aufeinanderfolgende Anführungszeichen allein entsprechen einer leeren Zeichenfolge. Beispiel:

```
%DEFINE empty=""
```

Die Variable *empty* enthält eine leere Zeichenfolge.

Wenn Ihre Variable Sonderzeichen wie Zeilenendezeichen enthält, verwenden Sie geschweifte Blockklammern um den Wert:

```
%DEFINE introduction={  
Hello,  
My name is John.  
%}
```

Sollen Anführungszeichen in einer Zeichenfolge verwendet werden, setzen Sie jeweils zwei Anführungszeichen hintereinander.

```
%DEFINE HI="say ""hello"""
```

Sie können auch geschweifte Blockklammern verwenden, um die Anführungszeichen zu umgehen:

```
%DEFINE HI={ say "hello" %}
```

Verwenden Sie einen DEFINE-Block, wenn Sie mehrere Variablen mit einer Anweisung DEFINE definieren wollen:

```
%DEFINE {  
    variable1="value1"  
    variable2="value2"  
    variable3="value3"  
    variable4="value4"  
%}
```

- **HTML-Formularbefehle: SELECT, INPUT und TEXTAREA**

Mit den HTML-Formularbefehlen SELECT, INPUT und TEXTAREA können Sie Variablen Werte zuordnen. Im folgenden Beispiel werden Standardbefehle für HTML-Formulare zum Definieren von Net.Data-Variablen verwendet:

```
<INPUT NAME="variable_name" TYPE=...>
```

oder

```
<SELECT NAME="variable_name">  
    <OPTION>value one  
    <OPTION>value two  
</SELECT>
```

Mit dem Befehl TEXTAREA können Sie eine Variable zuordnen, die sich auf mehrere Zeilen erstreckt oder Sonderzeichen wie Anführungszeichen enthält:

```
<TEXTAREA NAME="variable_name" ROWS="4">  
Please type the multi-line value  
of your variable here.  
</TEXTAREA>
```

variable_name ist der Name, den Sie der Variablen geben. Der Wert der Variablen wird durch die im Formular empfangene Eingabe bestimmt. In „HTML-Formulare“ auf Seite 39 finden Sie ein Beispiel dafür, wie diese Art der Variablendefinition in einem Net.Data-Makro verwendet wird.

- **Abfragezeichenfolgedaten**

Sie können über die Abfragezeichenfolge Variablen an Net.Data übergeben. Beispiel:

```
http://www.ibm.com/cgi-bin/db2www/stdqry1.d2w/input?field=custno
```

In diesem Beispiel geben der Variablenname *field* und der Variablenwert *custno* zusätzliche Daten an, die Net.Data über die Abfragezeichenfolge empfängt. Net.Data empfängt und verarbeitet die Daten so wie Formulardaten.

Verweisen auf Variablen

Sie können auf zuvor definierte Variablen verweisen, um an der Stelle des Verweises den Wert der Variablen zu erhalten. Ein Verweis auf eine Variable in Net.Data-Makros besteht aus dem Variablennamen, der in `$(` und `)` eingeschlossen ist. Beispiel:

```
$(variable_name)
$(homeURL)
```

Wenn Net.Data auf einen Variablenverweis trifft, ersetzt Net.Data den Variablenverweis durch den Wert der Variablen. Variablenverweise können Zeichenfolgen, andere Variablenverweise und Funktionsaufrufe enthalten.

Sie können Variablennamen dynamisch generieren. Bei diesem Verfahren können Sie mit Schleifen Tabellen unterschiedlicher Größe oder Eingabedaten für während der Laufzeit erstellte Listen verarbeiten, wenn die Anzahl in der Liste nicht im voraus ermittelt werden kann. Sie können z. B. Listen von HTML-Formularelementen generieren, die basierend auf von einer SQL-Abfrage zurückgegebenen Sätzen generiert werden.

Wenn Sie Variablen als Teil Ihrer Textdarstellungsanweisungen verwenden wollen, verweisen Sie in den HTML-Blöcken Ihres Makros darauf.

Ungültige Variablenverweise: Ungültige Variablenverweise werden als leere Zeichenfolge aufgelöst. Wenn z. B. ein Variablenverweis ungültige Zeichen wie ein Ausrufezeichen (!) enthält, wird der Verweis als leere Zeichenfolge aufgelöst .

Gültige Variablennamen müssen mit einem alphanumerischen Zeichen oder einem Unterstreichungszeichen anfangen und können aus alphanumerischen Zeichen einschließlich Punkt, Unterstreichungszeichen und Hash-Zeichen bestehen.

Beispiel 1: Variablenverweis in einer Programmverbindung (Link)

Sie definieren beispielsweise die Variable *homeURL* wie folgt:

```
%DEFINE homeURL="http://www.ibm.com/"
```

Dann können Sie Verweise auf die Home-Page in der Form `$(homeURL)` verwenden und eine Programmverbindung (Link) erstellen:

```
<A href="$(homeURL)">Home page</A>
```

Sie können in vielen Teilen des Net.Data-Makros auf Variablen verweisen. Überprüfen Sie die Sprachkonstrukte in diesem Kapitel, um zu ermitteln, in welchen Teilen des Makros Variablenverweise zulässig sind. Wenn die Variable zu dem Zeitpunkt, zu dem auf sie verwiesen wird, noch nicht definiert ist, gibt Net.Data eine leere Zeichenfolge zurück. Ein Variablenverweis allein definiert die Variable nicht.

Beispiel 2: Dynamisch generierte Variablenverweise

Angenommen, Sie führen eine SQL-Anweisung SELECT mit einer gewissen Anzahl von Elementen aus. Sie können ein HTML-Formular mit Eingabefeldern mit Hilfe der folgenden ROW-Blöcke erstellen:

```
...
%ROW {
<input type=text name=@dtw_rconcat("I", ROW_NUM) size=10 maxlength=10>
%}
...
```

Da Sie Eingabefelder erstellt haben, wollen Sie höchstwahrscheinlich auf die vom Benutzer eingegebenen Werte zugreifen, wenn das Formular zur Verarbeitung an Ihr Makro übergeben wird. Sie können eine Schleife codieren, um die Werte in einer Liste variabler Länge abzurufen:

```
<PRE>
...
@dtw_assign(rowIndex, "1")
%while (rowIndex <= rowCount) {
The value entered for row $(rowIndex) is: $(I$(rowIndex))
@dtw_add(rowIndex, "1", rowIndex) %}
...
</PRE>
```

Net.Data generiert zuerst mit dem Verweis I\$(rowIndex) den Variablennamen. Der erste Variablenname könnte z. B. I1 sein. Net.Data verwendet anschließend diesen Wert und löst damit den Variablenwert auf.

Beispiel 3: Ein Variablenverweis mit verschachtelten Variablenverweisen und einem Funktionsaufruf

```
%define my = "my"
%define u = "lower"
%define myLOWERvar = "hey"

$($ (my)@dtw_ruppercase(u)var)
```

Der Variablenverweis gibt den Wert von hey zurück.

Variablenarten

Sie können die folgenden Arten von Variablen in Ihren Makros verwenden:

- „Bedingungsvariablen“
- „Umgebungsvariablen“ auf Seite 58
- „Ausführbare Variablen“ auf Seite 58
- „Verdeckte Variablen“ auf Seite 60
- „Listenvariablen“ auf Seite 61
- „Tabellenvariablen“ auf Seite 62
- „Zusätzliche Variablen“ auf Seite 63
- „Variablen zur Tabellenverarbeitung“ auf Seite 63
- „Berichtsvariablen“ auf Seite 64
- „Sprachumgebungsvariablen“ auf Seite 65

Wenn Sie Variablen, die von Net.Data in spezieller Weise definiert werden (z. B. ENVVAR, LIST oder Bedingungslistenvariablen), Zeichenfolgen zuordnen, funktioniert die Variable nicht mehr in der definierten Weise. Das heißt, die Variable wird zu einer regulären Variablen, die eine Zeichenfolge enthält.

Informationen zur Syntax und Beispiele jeder Variable finden Sie im Handbuch *Net.Data Reference*.

Bedingungsvariablen

Bedingungsvariablen ermöglichen Ihnen, einen bedingten Wert für eine Variable mit Hilfe einer Methode zu definieren, die einem IF-THEN-Konstrukt ähnlich ist. Beim Definieren einer Bedingungsvariablen können Sie zwei mögliche Variablenwerte angeben. Wenn die erste Variable, auf die Sie verweisen, existiert, erhält die Bedingungsvariable den ersten Wert. Ansonsten erhält die Bedingungsvariable den zweiten Wert. Die Syntax für eine Bedingungsvariable sieht wie folgt aus:

```
varA = varB ? "value_1" : "value_2"
```

Wenn varB definiert ist, gilt varA="value_1", andernfalls gilt varA="value_2". Dies entspricht der Verwendung eines IF-Blocks wie im folgenden Beispiel:

```
%IF ($(varB))
    varA = "value_1"
%ELSE
    varA = "value_2"
%ENDIF
```

In „Listenvariablen“ auf Seite 61 finden Sie ein Beispiel für die Verwendung von Bedingungsvariablen mit Listenvariablen.

Umgebungsvariablen

Sie können auf Umgebungsvariablen verweisen, die der Web-Server dem Prozeß bzw. Thread, der Ihre Net.Data-Anforderung verarbeitet, zur Verfügung stellt. Wenn auf die Variable ENVVAR verwiesen wird, gibt Net.Data den aktuellen Wert der Umgebungsvariablen mit dem gleichen Namen zurück.

Die Syntax zur Definition von Umgebungsvariablen sieht folgendermaßen aus:

```
%DEFINE var=%ENVVAR
```

Dabei ist *var* der Name der Umgebungsvariablen, die definiert wird.

Zum Beispiel kann die Variable SERVER_NAME als Umgebungsvariable definiert werden:

```
%DEFINE SERVER_NAME=%ENVVAR
```

Ein Verweis auf sie könnte wie folgt aussehen:

```
The server is $(SERVER_NAME)
```

Die Ausgabe sähe wie folgt aus:

```
The server is www.software.ibm.com
```

Weitere Informationen zur Anweisung ENVVAR finden Sie im Handbuch *Net.Data Reference*.

Ausführbare Variablen

Sie können über einen Variablenverweis mit Hilfe ausführbarer Variablen andere Programme aufrufen.

Ausführbare Variablen werden in einem Net.Data-Makro mit Hilfe des Sprachkonstrukts EXEC im DEFINE-Block definiert. Weitere Informationen zum Sprachelement EXEC finden Sie im Kapitel über Sprachkonstrukte im Handbuch *Net.Data Reference*. Im folgenden Beispiel wird die Variable runit zur Ausführung des ausführbaren Programms testProg definiert:

```
%DEFINE runit=%EXEC "testProg"
```

Die Variable runit wird zu einer ausführbaren Variablen.

Net.Data führt das ausführbare Programm aus, wenn ein gültiger Variablenverweis in einem Net.Data-Makro erkannt wird. Zum Beispiel wird das Programm testProg ausgeführt, wenn in einem Net.Data-Makro ein gültiger Variablenverweis auf die Variable runit enthalten ist.

Eine einfache Methode besteht darin, auf eine ausführbare Variable aus einer anderen Variablendefinition heraus zu verweisen. Das folgende Beispiel illustriert diese Methode. Die Variable date wird als ausführbare Variable definiert. Anschließend wird dateRpt als Variablenverweis definiert, der die ausführbare Variable enthält.

```
%DEFINE date=%EXEC "date"  
%DEFINE dateRpt="Today is $(date)"
```

Für jedes Vorkommen des Variablenverweises \$(dateRpt) im Net.Data-Makro sucht Net.Data nach dem ausführbaren Programm date. Wenn das Programm gefunden wird, gibt Net.Data folgenden Wert zurück:

Today is Tue 11-07-1999

Wenn Net.Data eine ausführbare Variable in einem Makro feststellt, sucht Net.Data das angegebene ausführbare Programm nach folgender Methode:

1. Es durchsucht die Verzeichnisse, die in der Net.Data-Initialisierungsdatei durch EXEC_PATH definiert sind. Nähere Informationen finden Sie in „EXEC_PATH“ auf Seite 17.
2. Wenn Net.Data das Programm nicht findet, durchsucht das System die Verzeichnisse, die in der Umgebungsvariablen PATH bzw. in der Bibliothekenliste definiert sind. Wird das ausführbare Programm gefunden, führt Net.Data das Programm aus.

Einschränkung: Setzen Sie eine ausführbare Variable nicht auf den Wert der Ausgabe des aufgerufenen ausführbaren Programms. Im vorangegangenen Beispiel ist der Wert der Variablen date gleich NULL. Wenn Sie diese Variable in einem Funktionsaufruf DTW_ASSIGN verwenden, um ihren Wert einer anderen Variablen zuzuordnen, ist der Wert der neuen Variablen nach der Zuordnung ebenfalls NULL. Der einzige Zweck einer ausführbaren Variablen besteht darin, das Programm aufzurufen, das sie definiert.

Außerdem können Sie Parameter an das auszuführende Programm übergeben, indem Sie diese bei der Variablendefinition mit dem Programmnamen angeben. Im folgenden Beispiel werden die Werte für distance und time an das Programm *calcMPH* übergeben.

```
%DEFINE mph=%EXEC "calcMPH $(distance) $(time)"
```

Im nächsten Beispiel wird das Systemdatum als Teil des Berichts zurückgegeben:

```
%DEFINE database="celdial"
%DEFINE tstamp=%EXEC "date"

%FUNCTION(DTW_SQL) myQuery(){
SELECT CUSTNO, CUSTNAME from dist1.customer
  %REPORT{
    %ROW{
      <A HREF="/cgi-bin/db2www/exmp.d2w/report?value1=$(V1)&value2=$(V2)">
$(V1) $(V2) </A> <BR>
    %}
  %}
  %}

%HTML(report){
<H1>Report made: $(tstamp) </H1>
@myQuery()
%}
```

Jeder Bericht zeigt zur besseren Übersicht das Datum an. In diesem Beispiel werden außerdem die Kundennummer (CUSTNO) und der Kundenname (CUSTNAME) in eine Verbindung (Link) für ein anderes Net.Data-Makro eingefügt. Durch Anklicken eines Kunden im Bericht wird das Net.Data-Makro exmp.d2w aufgerufen, und die Nummer und der Name des Kunden werden an das Net.Data-Makro übergeben.

Verdeckte Variablen

Sie können verdeckte Variablen verwenden, um den tatsächlichen Namen einer Variablen für Anwendungsbenutzer unsichtbar zu machen, die die Quelle Ihrer Web-Seite mit ihrem Web-Browser anzeigen. Eine verdeckte Variable wird wie folgt definiert:

1. Definieren Sie eine Variable für jede Zeichenfolge, die Sie verdecken wollen, nach dem letzten Verweis auf die jeweilige Variable im HTML-Block. Variablen werden immer mit Hilfe des Sprachkonstrukts DEFINE definiert, nachdem sie im HTML-Block verwendet wurden, wie in folgendem Beispiel. Auf die Variablen `$$(variable)` wird zuerst verwiesen, anschließend werden sie definiert.
2. Verwenden Sie in dem HTML-Block, in dem auf die Variablen verwiesen wird, für einen Variablenverweis zwei Dollarzeichen anstelle eines einzelnen Dollarzeichens. Zum Beispiel `$(X)` anstelle von `$(X)`.

```
%HTML(INPUT){
<FORM ...>
<P>Select fields to view:
shanghai<SELECT NAME="Field">
<OPTION VALUE="$(name)"> Name
<OPTION VALUE="$(addr)"> Address
...
</FORM>
%}

%DEFINE {
name="customer.name"
addr="customer.address"
%}

%FUNCTION(DTW_SQL) mySelect() {
  SELECT $(Field) FROM customer
%}

...
```

Wenn ein Web-Browser das HTML-Formular anzeigt, werden `$(name)` und `$(addr)` durch `$(name)` bzw. `$(addr)` ersetzt, so daß die tatsächlichen Namen für Tabelle und Spalten im HTML-Formular nirgendwo vorkommen. Anwendungsbenutzer können nicht erkennen, daß die tatsächlichen Variablennamen verdeckt sind. Wenn der Benutzer das Formular übergibt, wird der Block HTML(REPORT) aufgerufen. Wenn @mySelect() den FUNCTION-Block aufruft, wird `$(Field)` in der SQL-Anweisung durch `customer.name` bzw. `customer.addr` in der SQL-Abfrage ersetzt.

Listenvariablen

Mit Listenvariablen können Sie eine begrenzte Wertefolge erstellen. Diese Variablenart ist besonders hilfreich beim Erstellen einer SQL-Abfrage mit mehreren Elementen, die in einigen WHERE- oder HAVING-Klauseln auftreten. Die Syntax für eine Listenvariable sieht wie folgt aus:

```
%LIST " value_separator " variable_name
```

Empfehlung: Leerzeichen sind signifikante Zeichen. Fügen Sie in den meisten Fällen ein Leerzeichen vor und nach dem Werttrennzeichen ein. Die meisten Abfragen verwenden boolesche oder mathematische Operatoren (z. B. AND, OR oder >) als Werttrennzeichen. Das folgende Beispiel zeigt die Verwendung von Bedingungsvariablen, verdeckten Variablen und Listenvariablen:

```
%HTML(INPUT){
<FORM METHOD="POST" ACTION="/cgi-bin/db2www/example2.d2w/report">
<H2>Select one or more cities:</H2>
<INPUT TYPE="checkbox" NAME="conditions" VALUE="$(cond1)">Sao Paolo<BR>
<INPUT TYPE="checkbox" NAME="conditions" VALUE="$(cond2)">Seattle<BR>
<INPUT TYPE="checkbox" NAME="conditions" VALUE="$(cond3)">Shanghai<BR>
<INPUT TYPE="submit" VALUE="Submit Query">
</FORM>
%}

%DEFINE{
DATABASE="custcity"
%LIST " OR " conditions
cond1="cond1='Sao Paolo'"
cond2="cond2='Seattle'"
cond3="cond3='Shanghai'"
whereClause= ? "WHERE $(conditions)" : ""
%}

%FUNCTION(DTW_SQL) mySelect() {
SELECT name, city FROM citylist
$(whereClause)
%}

%HTML(REPORT){
@mySelect()
%}
```

Wenn im HTML-Formular keine Kästchen ausgewählt werden, ist conditions gleich NULL, so daß whereClause in der Abfrage ebenfalls NULL ist. Andernfalls hat whereClause die durch OR getrennten Werte, die ausgewählt wurden. Wenn beispielsweise alle drei Städte ausgewählt werden, lautet die SQL-Abfrage:

```
SELECT name, city FROM citylist
WHERE cond1='Sao Paolo' OR cond2='Seattle' OR cond3='Shanghai'
```

Im folgenden wird Seattle ausgewählt, so daß die SQL-Abfrage wie folgt aussieht:

```
SELECT name, city FROM citylist
WHERE cond1='Seattle'
```

Tabellenvariablen

Die Tabellenvariable definiert eine Sammlung zusammengehöriger Daten. Sie enthält eine Gruppe von Zeilen und Spalten einschließlich einer Zeile mit Spaltenüberschriften. Eine Tabelle wird in einem Net.Data-Makro wie in der folgenden Anweisung definiert:

```
%DEFINE myTable=%TABLE(30)
```

Die Zahl nach %TABLE gibt die maximale Anzahl der Zeilen an, die diese Tabellenvariable enthalten kann. Wenn Sie eine Tabelle ohne Zeilenbegrenzung angeben wollen, müssen Sie den Standardwert bzw. ALL angeben:

```
%DEFINE myTable2=%TABLE  
%DEFINE myTable3=%TABLE(ALL)
```

Eine Tabelle hat bei ihrer Definition null Zeilen und null Spalten. Die einzige Möglichkeit, eine Tabelle mit Werten zu füllen, besteht darin, sie als Parameter OUT oder INOUT an eine Funktion zu übergeben oder die in Net.Data integrierten Tabellenfunktionen zu verwenden. Die Sprachumgebung DTW_SQL fügt die Ergebnisse einer Anweisung SELECT automatisch in eine Tabelle ein.

Bei Nicht-Datenbanksprachumgebungen, wie zum Beispiel DTW_REXX oder DTW_PERL, ist die Sprachumgebung auch dafür verantwortlich, die Tabellenwerte zu setzen. Die Prozedur bzw. das Programm der Sprachumgebung definiert die Tabellenwerte jedoch zellweise. Weitere Informationen dazu, wie Tabellenvariablen von Sprachumgebungen verwendet werden, finden Sie in „Verwenden der Sprachumgebungen“ auf Seite 91.

Sie können eine Tabelle zwischen Funktionen übergeben, indem Sie auf den Namen der Tabellenvariablen verweisen. Auf die einzelnen Elemente der Tabelle kann in einem REPORT-Block einer Funktion oder durch die Verwendung der Net.Data-Tabellenfunktionen verwiesen werden. Informationen zum Zugreifen auf einzelne Elemente in einer Tabelle innerhalb eines REPORT-Blocks finden Sie in „Variablen zur Tabellenverarbeitung“ auf Seite 63. Informationen zum Zugreifen auf einzelne Elemente einer Tabelle mit einer Tabellenfunktion finden Sie in „Tabellenfunktionen“ auf Seite 76. Tabellenvariablen werden in der Regel in einer SQL-Funktion mit Werten gefüllt und anschließend als Eingabe für einen Bericht entweder in der SQL-Funktion oder in einer anderen Funktion verwendet, nachdem sie als Parameter an diese Funktion übergeben wurden. Sie können Tabellenvariablen an eine beliebige Nicht-SQL-Funktion als Parameter IN, OUT oder INOUT übergeben. Tabellen können nur als Parameter OUT an SQL-Funktionen übergeben werden.

Wenn Sie auf eine Tabellenvariable verweisen, wird der Inhalt der Tabelle angezeigt und basierend auf der Einstellung der Variablen DTW_HTML_TABLE formatiert. Im folgenden Beispiel wird der Inhalt von myTable angezeigt:

```
%HTML (output) {  
    $(myTable)  
}
```

Die Spaltennamen und Feldwerte in einer Tabelle werden als Feldgruppenelemente mit dem Ursprung 1 angegeben.

Zusätzliche Variablen

Dies sind durch Net.Data definierte Variablen, die zu folgenden Zwecken verwendet werden können:

- Beeinflussen der Net.Data-Verarbeitung
- Ermitteln des Status eines Funktionsaufrufs
- Abrufen von Informationen zur Ergebnismenge einer Datenbankabfrage
- Bestimmen von Informationen zu Dateiadressen und Datumsangaben

Zusätzliche Variablen können entweder von Net.Data bestimmte vordefinierte Werte oder von Ihnen festgelegte Werte besitzen. Zum Beispiel bestimmt Net.Data den Wert der Variablen DTW_CURRENT_FILENAME nach der aktuellen Datei, die momentan verarbeitet wird, während Sie festlegen können, ob Net.Data zusätzliche, durch Tabulatoren und Zeilenvorschubzeichen verursachte Leerzeichen entfernen soll.

Vordefinierte Variablen werden innerhalb des Makros als Variablenverweise verwendet und liefern Informationen zum aktuellen Status von Dateien, Datumsangaben oder den Status eines Funktionsaufrufs. Sie könnten beispielsweise folgende Variable verwenden, um den Namen der aktuellen Datei abzurufen:

```
%REPORT {  
<p>This file is <i>$(DTW_CURRENT_FILENAME)</i>.</P>  
}
```

Änderbare Variablenwerte werden gewöhnlich mit Hilfe einer Anweisung DEFINE oder der Funktion @DTW_ASSIGN() festgelegt und geben Ihnen die Möglichkeit, die Verarbeitung des Makros durch Net.Data zu beeinflussen. Mit der folgenden Anweisung DEFINE könnten Sie zum Beispiel angeben, daß zusätzliche Leerzeichen (White Space) entfernt werden sollen:

```
%DEFINE DTW_REMOVE_WS="YES"
```

Variablen zur Tabellenverarbeitung

Net.Data definiert Variablen zur Tabellenverarbeitung, die in REPORT- und ROW-Blöcken verwendet werden können. Diese Variablen dienen zum Verweisen auf Werte aus SQL-Abfragen und Funktionsaufrufen.

Die Variablen zur Tabellenverarbeitung besitzen einen vordefinierten Wert, der von Net.Data festgelegt wird. Diese Variablen ermöglichen Ihnen, auf Werte aus den Ergebnismengen von SQL-Abfragen oder Funktionsaufrufen nach Spalte, Zeile oder Feld, die bzw. das verarbeitet wird, zu verweisen. Außerdem können Sie auf Informationen zur Anzahl der verarbeiteten Zeilen oder auf eine Liste aller Spaltennamen zugreifen.

Bei der Verarbeitung der Ergebnismenge aus einer SQL-Abfrage ordnet Net.Data zum Beispiel den Wert der Variablen Nn für jeden aktuellen Spaltennamen zu, so daß N1 der ersten Spalte, N2 der zweiten Spalte usw. zugeordnet wird. Sie können für Ihre Web-Seitenausgabe auf den aktuellen Spaltennamen verweisen.

Variablen zur Tabellenverarbeitung werden als Variablenverweise innerhalb des Makros verwendet. Zum Beispiel können Sie den Namen der aktuellen Spalte, die verarbeitet wird, folgendermaßen abrufen:

```
%REPORT {  
<p>Column 1 is <i>$(N1)</i>.</P>  
}
```

Variablen zur Tabellenverarbeitung liefern außerdem Informationen zu den Ergebnissen einer Abfrage. Sie können im Makro auf die Variable TOTAL_ROWS verweisen, um die Anzahl der von einer SQL-Abfrage zurückgegebenen Zeilen abzufragen, wie im folgenden Beispiel gezeigt:

```
Names found: $(TOTAL_ROWS)
```

Einige der Variablen zur Tabellenverarbeitung werden von anderen Variablen oder integrierten Funktionen beeinflusst. Zum Beispiel setzt die Variable TOTAL_ROWS voraus, daß die SQL-Sprachumgebungsvariable DTW_SET_TOTAL_ROWS aktiviert ist, so daß Net.Data den Wert von TOTAL_ROWS zuordnet, wenn die Ergebnisse aus einer SQL-Abfrage oder einem Funktionsaufruf verarbeitet werden, wie in folgendem Beispiel gezeigt wird:

```
%DEFINE DTW_SET_TOTAL_ROWS="YES"  
...
```

```
Names found: $(TOTAL_ROWS)
```

Berichtsvariablen

Net.Data zeigt Web-Seitenausgaben, die durch das Makro generiert werden, in einem Standardberichtsformat an. Das Standardberichtsformat wird in einem Tabellenformat mit Hilfe von <PRE> </PRE> angezeigt. Sie können den Standardbericht außer Kraft setzen, indem Sie einen REPORT-Block mit Anweisungen zum Anzeigen der Ausgabe definieren oder eine der Berichtsvariablen verwenden, um die Generierung des Standardberichts zu verhindern.

Berichtsvariablen unterstützen Sie bei der Anpassung der Anzeige der Web-Seitenausgabe und bei der Verwendung der Web-Seitenausgabe mit Standardberichten und Net.Data-Tabellen. Diese Variablen müssen vor ihrer Verwendung mit Hilfe einer DEFINE-Anweisung oder der Funktion @DTW_ASSIGN() definiert werden.

Die Berichtsvariablen definieren die Verwendung von Leerzeichen, überschreiben Standardberichtsformate, legen HTML-Tabellenausgaben oder Standardtabellenausgaben fest und bestimmen weitere Anzeigemerkmale. Zum Beispiel können Sie mit der Variablen ALIGN die Verwendung führender und nachgestellter Leerzeichen für die Variablen zur Tabellenverarbeitung steuern. Im folgenden Beispiel wird die Variable ALIGN verwendet, um jeden Spaltennamen in einer Liste, die durch eine Abfrage zurückgegeben wird, durch Leerzeichen zu trennen.

```
%DEFINE ALIGN="YES"
...
<p>Your query was on these columns: $(NLIST)
```

Mit der Berichtsvariablen START_ROW_NUM können Sie festlegen, ab welcher Zeile die Ergebnisse einer Abfrage angezeigt werden sollen. Zum Beispiel gibt der folgende Variablenwert an, daß Net.Data die Ergebnisse einer Abfrage ab der dritten Zeile anzeigen soll:

```
%DEFINE START_ROW_NUM = "3"
```

Sie können außerdem festlegen, ob Net.Data HTML-Befehle für die Standardformatierung verwenden soll.

Wenn der Wert der Variablen DTW_HTML_TABLE auf YES gesetzt ist, wird keine Tabelle mit formatiertem Text erstellt, sondern eine HTML-Tabelle.

```
%DEFINE DTW_HTML_TABLE="YES"

%FUNCTION(DTW_SQL){
SELECT NAME, ADDRESS FROM $(qTable)
%}
```

Sprachumgebungsvariablen

Diese Variablen werden mit Sprachumgebungen verwendet und beeinflussen die Verarbeitung einer Anforderung durch die Sprachumgebung.

Mit diesen Variablen können Sie Aufgaben wie das Herstellen von Verbindungen zu Datenbanken, das Aktivieren von Sprachenunterstützung und das Feststellen der erfolgreichen Ausführung einer SQL-Anweisung durchführen.

Zum Beispiel können Sie mit der Variablen SQL_STATE auf den von der Datenbank zurückgegebenen SQLSTATE-Wert zugreifen bzw. diesen Wert anzeigen.

```
%FUNCTION (DTW_SQL) val1() {
  select * from customer
%REPORT {
  ...
%ROW {
  ...
%}
  SQLSTATE=$(SQL_STATE)
%}
```

Net.Data-Funktionen

Net.Data stellt integrierte Funktionen zur Verwendung in Ihren Anwendungen bereit, wie Funktionen zur Wort- und Zeichenfolgebearbeitung und Funktionen, mit denen Tabellenvariablenfunktionen abgerufen und festgelegt werden. Sie können auch Funktionen zur Verwendung mit Ihrer Anwendung definieren, z. B. zum Aufrufen eines externen Programms oder einer gespeicherten Prozedur.

Benutzerdefinierte Funktionen

Diese Funktionen, die Sie zur Verwendung mit Ihrer Anwendung definieren, rufen z. B. ein externes Programm oder eine gespeicherte Prozedur auf.

Integrierte Net.Data-Funktionen

Die Funktionen, die von Net.Data zur Verwendung in Ihren Anwendungen bereitgestellt werden, etwa Funktionen zur Wort- und Zeichenfolgebearbeitung und Funktionen, mit denen Tabellenvariablen abgerufen und festgelegt werden.

In diesen Abschnitten werden die folgenden Themen beschrieben:

- „Definieren von Funktionen“
- „Aufrufen von Funktionen“ auf Seite 72
- „Aufrufen von integrierten Net.Data-Funktionen“ auf Seite 72

Definieren von Funktionen

Verwenden Sie zum Definieren eigener Funktionen im Makro einen FUNCTION-Block oder einen MACRO_FUNCTION-Block:

FUNCTION-Block

Definiert eine Unteroutine, die von einem Net.Data-Makro aufgerufen wird und von einer Sprachumgebung verarbeitet wird. FUNCTION-Blöcke müssen Sprachanweisungen oder Aufrufe an ein externes Programm enthalten.

MACRO_FUNCTION-Block

Definiert eine Unteroutine, die von einem Net.Data-Makro aufgerufen wird und von Net.Data anstatt von einer Sprachumgebung verarbeitet wird. MACRO_FUNCTION-Blöcke können beliebige, in einem HTML-Block zulässige Anweisungen enthalten.

Syntax: Verwenden Sie die folgende Syntax zur Definition von Funktionen:

FUNCTION-Block:

```
%FUNCTION(type) function-name([usage] [datatype] parameter, ...)  
[RETURNS(return-var)] {  
    executable-statements  
    [report-block]  
    ...  
    [report-block]  
    [message-block]  
%}
```

MACRO_FUNCTION-Block:

```
%MACRO_FUNCTION function-name([usage] parameter, ...) [RETURNS(return-var)] {  
    executable-statements  
    report-block  
    ...  
    report-block  
%}
```

Dabei gilt folgendes:

type Gibt eine Sprachumgebung an, die in der Initialisierungsdatei konfiguriert wird. Die Sprachumgebung ruft einen speziellen Sprachprozessor auf (der die ausführbaren Anweisungen verarbeitet) und stellt eine Standardschnittstelle zwischen Net.Data und dem Sprachprozessor bereit.

function-name

Gibt den Namen des FUNCTION- oder MACRO_FUNCTION-Blocks an. Ein Funktionsaufruf gibt *function-name* mit einem vorangestellten kommerziellen A (@) an. Nähere Informationen finden Sie in „Aufrufen von Funktionen“ auf Seite 72.

Sie können mehrere FUNCTION-oder MACRO_FUNCTION-Blöcke mit dem gleichen Namen definieren, so daß sie gleichzeitig verarbeitet werden. Jeder der Blöcke muß eine identische Parameterliste besitzen. Wenn Net.Data die Funktion aufruft, werden alle gleichnamigen FUNCTION-Blöcke bzw. gleichnamigen MACRO_FUNCTION-Blöcke in der Reihenfolge ausgeführt, in der sie im Net.Data-Makro definiert sind.

syntax Gibt an, ob ein Parameter ein Eingabeparameter (IN), ein Ausgabeparameter (OUT) oder beides (INOUT) ist. Diese Angabe bestimmt, ob der Parameter an den FUNCTION-Block bzw. den MACRO_FUNCTION-Block übergeben und/oder von ihm empfangen wird. Die Verwendungsart gilt für alle nachfolgenden Parameter in der Parameterliste, bis sie durch eine andere Verwendungsart geändert wird. Die Standardart ist IN.

datatype

Der Datentyp des Parameters. Einige Sprachumgebungen erwarten Datentypen für die übergebenen Parameter. Die SQL-Sprachumgebung z. B. erwartet sie beim Aufrufen gespeicherter Prozeduren ebenso wie die Direktaufrufsprachumgebung beim Aufrufen von Programmen. Weitere Informationen zu den unterstützten Datentypen für die verwendete Sprachumgebung finden Sie in „Verwenden der Sprachumgebungen“ auf Seite 91.

parameter

Der Name einer Variablen mit lokalem Geltungsbereich, die durch den Wert eines entsprechenden in einem Funktionsaufruf angegebenen Arguments ersetzt wird. Parameterverweise, z. B. \$(*parm1*), in den ausführbaren Anweisungen oder REPORT-Blöcken werden durch den tatsächlichen Wert des Parameters ersetzt. Zudem werden die Parameter an die Sprachumgebung übergeben. Ausführbare Anweisungen, die die spezifische Syntax dieser Sprache verwenden, können darauf zugreifen oder sie als Umgebungsvariablen verwenden. Verweise auf Parametervariablen sind außerhalb der FUNCTION- bzw. MACRO_FUNCTION-Blöcke ungültig.

return-var

Geben Sie diesen Parameter nach dem Schlüsselwort RETURNS an, um einen speziellen OUT-Parameter zu definieren. Der Wert der Rückkehrvariablen wird im Funktionsblock zugeordnet, und sein Wert wird an der Stelle im Makro zurückgegeben, von der die Funktion aufgerufen wurde. Zum Beispiel wird im folgenden Satz `<p>My name is @my_name()`. die Angabe `@my_name()` durch den Wert der Rückkehrvariablen ersetzt. Wenn Sie die Klausel RETURNS nicht angeben, hat der Funktionsaufruf einen der folgenden Werte:

- NULL, falls der Rückkehrcode aus dem Aufruf an die Sprachumgebung Null ist
- Den Wert des Rückkehrcodes, wenn der Rückkehrcode ungleich Null ist

executable-statements

Die Gruppe der Sprachanweisungen, die an die angegebene Sprachumgebung zur Verarbeitung übergeben wird, nachdem die Variablen ersetzt und die Funktionen verarbeitet wurden. *executable-statements* können Net.Data-Variablenverweise und Net.Data-Funktionsaufrufe enthalten.

Bei FUNCTION-Blöcken ersetzt Net.Data alle Variablenverweise durch die Variablenwerte, führt alle Funktionsaufrufe aus und ersetzt die Funktionsaufrufe mit den sich jeweils ergebenden Werten, bevor die ausführbaren Anweisungen an die Sprachumgebung übergeben werden. Jede Sprachumgebung verarbeitet die Anweisungen auf andere Weise. Weitere Informationen zur Angabe ausführbarer Anweisungen oder zum Aufrufen ausführbarer Programme finden Sie in „Ausführbare Variablen“ auf Seite 58.

Bei MACRO_FUNCTION-Blöcken sind die ausführbaren Anweisungen eine Kombination aus Text und Net.Data-Makrosprachkonstrukten. In diesem Fall spielt die Sprachumgebung keine Rolle, weil Net.Data als Programmiersprachenprozessor fungiert und die ausführbaren Anweisungen verarbeitet.

report-block

Definiert einen oder mehrere REPORT-Blöcke zur Behandlung der Ausgabe des FUNCTION- oder MACRO_FUNCTION-Blocks. Siehe „REPORT-Blöcke“ auf Seite 79.

message-block

Definiert den MESSAGE-Block, der alle vom FUNCTION-Block zurückgemeldeten Nachrichten verarbeitet. Siehe „MESSAGE-Blöcke“ auf Seite 70.

Definieren Sie Funktionen außerhalb eines anderen Blocks und bevor diese im Net.Data-Makro aufgerufen werden.

Verwenden von Sonderzeichen in Funktionen

Wenn Zeichen, die der Syntax der Net.Data-Sprachkonstrukte entsprechen, in für die Sprachanweisungen eines FUNCTION-Blocks als Teil eines syntaktisch gültigen, eingebetteten Programmcodes (wie für REXX oder Perl) verwendet werden, können sie fälschlicherweise als Net.Data-Sprachkonstrukte interpretiert werden und so Fehler oder unvorhersehbare Ergebnisse in einem Makro verursachen.

Zum Beispiel könnte eine Perl-Funktion die Begrenzungszeichen für COMMENT-Blöcke (%) verwenden. Bei der Ausführung des Makros werden die Zeichen %{} als Anfang eines COMMENT-Blocks interpretiert. Net.Data sucht dann nach dem Ende des COMMENT-Blocks, das von Net.Data am Ende des FUNCTION-Blocks erwartet wird. Net.Data sucht dann nach dem Ende des FUNCTION-Blocks und gibt, wenn das Ende nicht gefunden wird, einen Fehler aus.

Mit einer der folgenden Methoden können Sie die Begrenzungszeichen für COMMENT-Blöcke sowie alle anderen Net.Data-Sonderzeichen als Teil Ihres eingebetteten Programmcodes verwenden, ohne daß sie von Net.Data als Sonderzeichen interpretiert werden:

- Verwenden Sie die Anweisung EXEC zum Aufrufen des Programmcodes, anstatt den Code inline einzufügen.
- Verwenden Sie einen Variablenverweis zur Angabe der Sonderzeichen.

Die folgende Perl-Funktion zum Beispiel enthält als Teil ihrer Perl-Sprachanweisungen Zeichen, die eine COMMENT-Blockbegrenzung, %{}, darstellen:

```
%FUNCTION(DTW_PERL) func() {  
    ...  
    for $num_words (sort bynumber keys %{} $Rtitles{$num} }) {  
        &make_links($Rtitles{$num}{$num_words});  
    }  
    ...  
%}
```

Um sicherzustellen, daß Net.Data die Zeichen %{} als Perl-Quellencode und nicht als eine COMMENT-Blockbegrenzung von Net.Data interpretiert, sollte die Funktion auf eine der folgenden Weisen umgeschrieben werden:

- Mit der Anweisung %EXEC:

```
%FUNCTION(DTW_PERL) func() {  
    %EXEC{ func.pr1 %}  
%}
```

- Mit einem Variablenverweis zur Angabe der Zeichen %{}:

```
%define percent_openbrace = "%{"  
  
%FUNCTION(DTW_PERL) func() {  
    ...  
    for $num_words (sort by number keys ${percent_openbrace} $Rtitles{$num} ) {  
        &make_links($Rtitles{$num}{$num_words});  
    }  
    ...  
%}
```

MESSAGE-Blöcke

Anhand des MESSAGE-Blocks können Sie die weitere Vorgehensweise festlegen, nachdem ein Funktionsaufruf erfolgreich ausgeführt wurde bzw. fehlgeschlagen ist, und Informationen für den Aufrufenden der Funktion anzeigen. Beim Verarbeiten einer Nachricht setzt Net.Data die Sprachumgebungsvariable RETURN_CODE für jeden Funktionsaufruf auf einen FUNCTION-Block. RETURN_CODE wird bei einem Funktionsaufruf nicht auf einen MACRO_FUNCTION-Block gesetzt.

Ein MESSAGE-Block besteht aus einer Reihe von Nachrichtenanweisungen, von denen jede einen Rückkehrcodewert, einen Nachrichtentext und eine durchzuführende Aktion definiert. Die Syntax eines MESSAGE-Blocks wird im Kapitel über die Sprachkonstrukte im Handbuch *Net.Data Reference* gezeigt.

Ein MESSAGE-Block kann für einen globalen oder lokalen Bereich gelten. Wenn der MESSAGE-Block in einem FUNCTION-Block definiert ist, ist sein Geltungsbereich für diesen FUNCTION-Block lokal. Wenn er in der äußeren Makroebene angegeben wird, hat der MESSAGE-Block einen globalen Geltungsbereich und ist für alle im Net.Data-Makro ausgeführten Funktionsaufrufe aktiv. Wenn Sie mehrere globale MESSAGE-Blöcke definieren, ist der zuletzt definierte Block aktiv.

Net.Data verwendet die folgenden Regeln zur Verarbeitung des Werts der Variablen RETURN_CODE von einem Funktionsaufruf:

1. Ein lokaler MESSAGE-Block wird auf eine exakte Übereinstimmung hin überprüft, und die Verarbeitung wird je nach Angabe beendet (exit) oder fortgesetzt (continue).
2. Wenn RETURN_CODE ungleich 0 ist, wird ein lokaler MESSAGE-Block auf +default bzw. -default hin überprüft, und die Verarbeitung wird abhängig vom Vorzeichen von RETURN_CODE je nach Angabe beendet oder fortgesetzt.
3. Wenn RETURN_CODE ungleich 0 ist, wird ein lokaler MESSAGE-Block auf default hin überprüft, und die Verarbeitung je nach Angabe beendet oder fortgesetzt.
4. Ein globaler MESSAGE-Block wird auf eine exakte Übereinstimmung hin überprüft, und die Verarbeitung wird je nach Angabe beendet oder fortgesetzt.
5. Wenn RETURN_CODE ungleich 0 ist, wird ein globaler MESSAGE-Block auf +default bzw. -default hin überprüft, und die Verarbeitung wird abhängig vom Vorzeichen von RETURN_CODE je nach Angabe beendet oder fortgesetzt.
6. Wenn RETURN_CODE ungleich 0 ist, wird ein globaler MESSAGE-Block auf default hin überprüft, und die Verarbeitung je nach Angabe beendet oder fortgesetzt.
7. Wenn RETURN_CODE ungleich 0 ist, gibt Net.Data die interne Standardnachricht aus und beendet die Verarbeitung.

Das folgende Beispiel zeigt einen Teil eines Net.Data-Makros mit einem globalen MESSAGE-Block und einem MESSAGE-Block für eine Funktion:

```
%{ global message block %}
%MESSAGE {
    -100      : "Return code -100 message"    : exit
    100       : "Return code 100 message"     : continue
    +default : {
This is a long message that spans more
than one line. You can use HTML tags, including
links and forms, in this message. %}    : continue
%}

%{ local message block inside a FUNCTION block %}
%FUNCTION(DTW_REXX) my_function() {
    %EXEC { my_command.mbr %}
%MESSAGE {
    -100      : "Return code -100 message"    : exit
    100       : "Return code 100 message"     : continue
    -default : {
This is a long message that spans more
than one line. You can use HTML tags, including
links and forms, in this message. %}    : exit
%}
```

Wenn *my_function()* mit einem RETURN_CODE-Wert von 50 beendet wird, verarbeitet Net.Data den Fehler in folgender Reihenfolge:

1. Es wird überprüft, ob es eine exakte Übereinstimmung im lokalen MESSAGE-Block vorhanden ist.
2. Es wird überprüft, ob +default im lokalen MESSAGE-Block vorhanden ist.
3. Es wird überprüft, ob default im lokalen MESSAGE-Block vorhanden ist.
4. Es wird überprüft, ob eine exakte Übereinstimmung im globalen MESSAGE-Block vorhanden ist.
5. Es wird überprüft, ob +default im globalen MESSAGE-Block vorhanden ist.

Wird eine Übereinstimmung gefunden, sendet Net.Data den Nachrichtentext an den Web-Browser und überprüft die angeforderte Aktion.

Wenn Sie continue angeben, setzt Net.Data die Verarbeitung des Net.Data-Makros fort, nachdem der Nachrichtentext angezeigt wurde. Angenommen, ein Makro ruft *my_functions()* fünf Mal auf und bei der Verarbeitung mit dem MESSAGE-Block aus obigem Beispiel wird der Fehler 100 gefunden. In diesem Fall könnte die Ausgabe des Programms folgendermaßen aussehen:

```
.
.
.
11 May 1997                $245.45
13 May 1997                $623.23
19 May 1997                $ 83.02
    return code 100 message
22 May 1997                $ 42.67

Total:                    $994.37
```

Aufrufen von Funktionen

Verwenden Sie eine Net.Data-Funktionsaufrufanweisung, um benutzerdefinierte oder integrierte Funktionen aufzurufen. Verwenden Sie das kommerzielle A (@), gefolgt von einem Funktionsnamen bzw. Makrofunktionsnamen:

```
@function_name([ argument,... ])
```

function_name

Dies ist der Name der aufzurufenden Funktion bzw. Makrofunktion. Die Funktion muß bereits im Net.Data-Makro definiert sein, es sei denn, es handelt sich um eine integrierte Funktion.

argument Dies ist der Name einer Variablen, eine Zeichenfolge in Anführungszeichen, ein Variablenverweis oder ein Funktionsaufruf. Die Argumente in einem Funktionsaufruf werden mit den Parametern in einer Funktions- oder Makrofunktionsparameterliste abgeglichen. Außerdem wird jedem Parameter bei der Verarbeitung der Funktion bzw. Makrofunktion der Wert des entsprechenden Arguments zugeordnet. Die Argumente müssen dieselbe Anzahl und Art wie die zugehörigen Parameter aufweisen.

Zeichenfolgen in Anführungszeichen als Argumente können Variablenverweise und Funktionsaufrufe enthalten.

Beispiel 1: Funktionsaufruf mit einem Zeichenfolgeargument

```
@myFunction("abc")
```

Beispiel 2: Funktionsaufruf mit einer Variablen und Funktionsaufrufargumenten

```
@myFunction(myvar, @DTW_rADD("2","3"))
```

Beispiel 3: Funktionsaufruf mit einem Zeichenfolgeargument, das einen Variablenverweis und einen Funktionsaufruf enthält

```
@myFunction("abc$(myvar)def@DTW_rADD("2","3")ghi")
```

Aufrufen von integrierten Net.Data-Funktionen

Net.Data verfügt über zahlreiche integrierte Funktionen, die die Entwicklung von Web-Seiten vereinfachen. Diese Funktionen sind von Net.Data bereits definiert, d. h. Sie brauchen sie nicht mehr zu definieren. Sie können diese Funktionen so wie andere Funktionen aufrufen.

Abb. 7 auf Seite 73 zeigt die Interaktion zwischen den integrierten Net.Data-Funktionen und dem Makro.

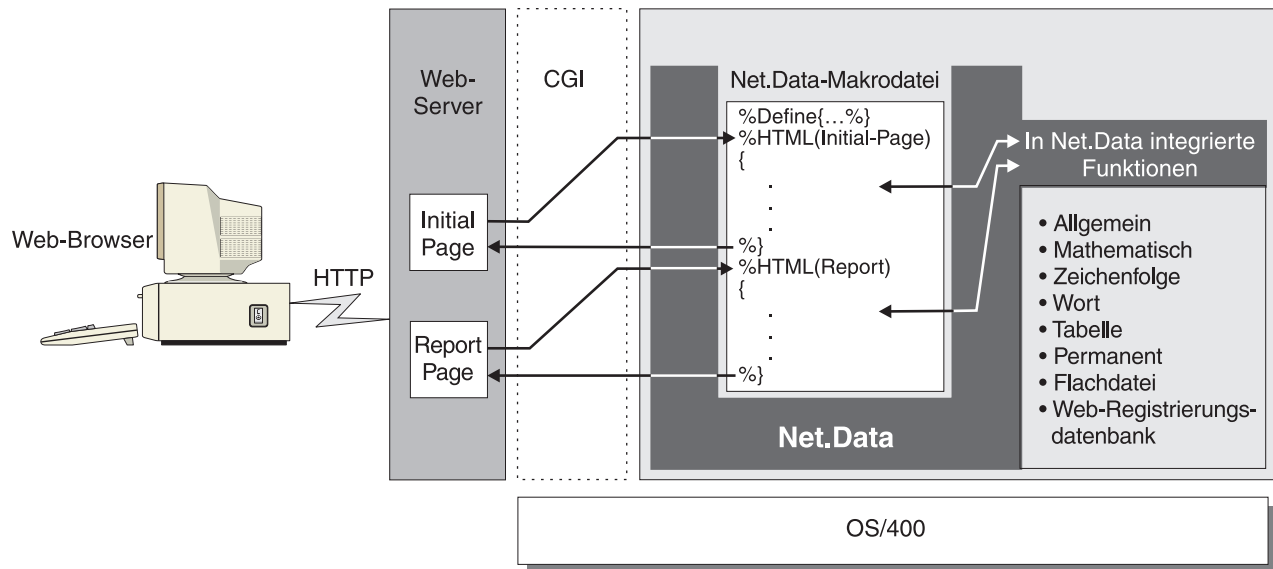


Abbildung 7. In Net.Data integrierte Funktionen

Es gibt drei Methoden, mit denen integrierte Funktionen ihre Ergebnisse zurückgeben können. Dies hängt vom Präfix ab.

- **DTW_, DTWF_ und DTWR_:** Die Ergebnisse des Aufrufs werden in einem Ausgabeparameter zurückgegeben, oder es wird kein Ergebnis zurückgegeben. (**DTWF_** ist das Präfix für Funktionen von unstrukturierten Textdateien. **DTWR_** ist das Präfix von Funktionen für Web-Registrierdatenbanken.)
- **DTW_r, DTWF_r und DTWR_r:** Die Ergebnisse des Funktionsaufrufs ersetzen den Funktionsaufruf im Makro. Ebenso ersetzt der Wert des Schlüsselworts RETURNS den Funktionsaufruf für eine benutzerdefinierte Funktion, in der das Schlüsselwort RETURNS angegeben ist.
- **DTW_m:** Mehrere Ergebnisse werden in allen Parametern zurückgegeben, die an die Funktion übergeben werden.

Einige integrierte Funktionen unterstützen nicht jede Art. Wenn Sie ermitteln möchten, um welche Art es sich bei einer bestimmten integrierten Funktion handelt, ziehen Sie das Kapitel zu den integrierten Net.Data-Funktionen im Handbuch *Net.Data Reference* zu Rate.

In den folgenden Abschnitten wird eine allgemeine Übersicht über die integrierten Net.Data-Funktionen gegeben. Mit diesen Funktionen können Sie allgemeine und mathematische Funktionen sowie Zeichenfolge-, Wort- bzw. Tabellenbearbeitungsfunktionen ausführen. Außerdem können Sie permanente Funktionen für die Transaktionsverarbeitung verwenden. Beschreibungen der einzelnen Funktionen mit Syntax und Beispielen finden Sie im Handbuch *Net.Data Reference*. Für einige dieser Funktionen müssen Variablen definiert werden, bevor sie verwendet werden können, oder sie müssen in einem spezifischen Kontext verwendet werden. Nicht alle Betriebssysteme unterstützen jede integrierte Funktion.

Sie können anhand des Handbuchs *Net.Data Reference* ermitteln, welche Funktionen für Ihr Betriebssystem unterstützt werden.

- „Allgemeine Funktionen“
- „Mathematische Funktionen“ auf Seite 75
- „Zeichenfolgefunktionen“ auf Seite 75
- „Wortfunktionen“ auf Seite 75
- „Tabellenfunktionen“ auf Seite 76
- „Funktionen für unstrukturierte Textdateien“ auf Seite 76
- „Funktionen für Web-Registrierdatenbanken“ auf Seite 76
- „Permanente Funktionen“ auf Seite 77

Allgemeine Funktionen

Mit Hilfe dieser Gruppe von Funktionen können Sie Web-Seiten durch Ändern von Daten oder Zugreifen auf Systemservices entwickeln. Damit können Sie E-Mail senden, HTTP-Cookies verarbeiten, HTML-Escape-Codes generieren und andere nützliche Informationen vom System abrufen.

Sie verwenden zum Beispiel die Funktion DTW_EXIT, um festzulegen, daß Net.Data ein Makro verlassen soll, ohne die restlichen Anweisungen des Makros zu verarbeiten, wenn eine bestimmte Bedingung eintritt:

```
%HTML(cache_example) {  
  
<html>  
<head>  
  <title>This is the page title</title>  
</head>  
<body>  
  <center>  
    <h3>This is the Main Heading</h3>  
    <!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!>  
    <! Joe Smith sees a very short page                               !>  
    <!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!>  
    %IF (customer == "Joe Smith")  
  </body>  
</html>  
  
@DTW_EXIT()  
  
%ENDIF  
  
...  
  
</body>  
</html>  
%}
```

Eine weitere nützliche Funktion ist die Funktion DTW_URLESCSEQ, die Zeichen, die in einer URL-Adresse nicht zulässig sind, durch die entsprechenden Escape-Werte ersetzt. Wenn beispielsweise die Eingabevariable string1 den Inhalt "Guys & Dolls" hat, ordnet DTW_URLESCSEQ die Ausgabevariable dem Wert "Guys%20%26%20Dolls" zu.

Mathematische Funktionen

Diese Funktionen führen mathematische Operationen aus, über die Sie numerische Daten berechnen und ändern können. Neben den mathematischen Standardoperationen können auch Modulus-Divisionen ausgeführt, eine Ergebnissenauigkeit angegeben und Exponentialschreibweise verwendet werden.

Zum Beispiel potenziert die Funktion DTW_POWER den Wert des ersten Parameters mit dem Wert des zweiten Parameters und gibt das Ergebnis zurück. Siehe dazu das folgende Beispiel:

```
@DTW_POWER("2", "-3", result)
```

DTW_POWER gibt ".125" in der Variablen result zurück.

Zeichenfolgefunktionen

Diese Funktionen können zum Bearbeiten von Zeichen in Zeichenfolgen verwendet werden. So können Sie zum Beispiel eine Zeichenfolge von Klein- in Großschreibung (oder umgekehrt) umsetzen, Zeichen einfügen oder löschen, einen Zeichenfolgewart einer anderen Variablen zuordnen und noch andere nützliche Funktionen ausführen.

Zum Beispiel können Sie DTW_ASSIGN verwenden, um den Wert einer Eingabevariablen einer Ausgabevariablen zuzuordnen. Sie können diese Funktion auch verwenden, um eine Variable in einem Makro zu verwenden. Im folgenden Beispiel wird der Variablen RC Null zugeordnet.

```
@DTW_ASSIGN(RC, "0")
```

Weitere Zeichenfolgefunktionen sind DTW_CONCAT zum Verknüpfen von Zeichenfolgen und DTW_INSERT zum Einfügen von Zeichenfolgen an einer bestimmten Position sowie viele andere Funktionen zum Bearbeiten von Zeichenfolgen.

Wortfunktionen

Diese Funktionen können zum Bearbeiten von Wörtern in Zeichenfolgen verwendet werden. Die meisten dieser Funktionen arbeiten auf ähnliche Weise wie Zeichenfolgefunktionen, jedoch bezogen auf ganze Wörter. Hiermit können Sie zum Beispiel die Anzahl der Wörter in einer Zeichenfolge zählen, Wörter entfernen oder nach einem Wort in einer Zeichenfolge suchen.

Verwenden Sie beispielsweise DTW_DELWORD, um eine bestimmte Anzahl von Wörtern aus einer Zeichenfolge zu löschen:

```
@DTW_DELWORD("Now is the time", "2", "2", result)
```

DTW_DELWORD gibt die Zeichenfolge "Now time" zurück.

Weitere Wortfunktionen sind DTW_WORDLENGTH (gibt die Anzahl der Zeichen in einem Wort zurück) und DTW_WORDPOS (gibt die Position eines Worts in einer Zeichenfolge zurück).

Tabellenfunktionen

Diese Funktionen können zum Generieren von Berichten oder Formularen mit Hilfe der Daten in einer Net.Data-Tabellenvariable verwendet werden. Mit diesen Funktionen können Sie auch Net.Data-Tabellen erstellen sowie Werte in diesen Tabellen bearbeiten und abrufen. Tabellenvariablen enthalten eine Gruppe von Werten und ihre zugehörigen Spaltennamen. Sie bieten eine bequeme Möglichkeit, ganze Wertegruppen an eine Funktion weiterzugeben.

Zum Beispiel fügt DTW_TB_APPENDROW eine Zeile an die Tabelle an.

Im folgenden Beispiel fügt Net.Data zehn Zeilen an die Tabelle myTable an:

```
@DTW_TB_APPENDROW(myTable, "10")
```

Außerdem gibt DTW_TB_DUMP den Inhalt einer Makrotabellenvariablen zurück, die in die Befehle <PRE></PRE> eingeschlossen ist, wobei jede Zeile der Tabelle in einer anderen Zeile angezeigt wird. DTW_TB_CHECKBOX gibt einen oder mehrere HTML-Markierungsfeldeingabefehle aus einer Makrotabellenvariablen zurück.

Funktionen für unstrukturierte Textdateien

Die FFI-Schnittstelle (Flat File Interface - Schnittstelle für unstrukturierte Textdateien) kann zum Öffnen, Lesen und Bearbeiten von Daten aus unstrukturierten Textdateiquellen (Textdateien) sowie zum Speichern von Daten in unstrukturierte Textdateien verwendet werden.

Zum Beispiel schreibt DTWF_APPEND den Inhalt einer Tabellenvariablen an das Ende einer Datei, und DTWF_DELETE löscht Sätze aus einer Datei.

Außerdem unterstützen die FFI-Funktionen Dateisperren mit DTWF_CLOSE und DTWF_OPEN. DTWF_OPEN sperrt eine Datei, damit sie eine andere Anforderung weder lesen noch aktualisieren kann. DTWF_CLOSE gibt die Datei frei, wenn Net.Data sie nicht länger benötigt, damit andere Anforderungen auf die Datei zugreifen können.

Funktionen für Web-Registrierdatenbanken

Die Funktionen für Web-Registrierdatenbanken können zum Verwalten der Registrierdatenbanken und der darin enthaltenen Einträge verwendet werden. Eine Web-Registrierdatenbank ist eine Datei mit einem von Net.Data verwalteten Schlüssel, die das einfache Hinzufügen, Abrufen und Löschen von Einträgen ermöglicht.

Zum Beispiel fügt DTWR_ADDENTRY Einträge hinzu, während DTWR_DELENTY Einträge löscht. DTWR_LISTSUB gibt Informationen zu den Registrierdatenbankeinträgen in einem OUT-Tabellenparameter zurück, und DTWR_UPDATEENTRY ersetzt vorhandene Werte für einen angegebenen Registrierdatenbankeintrag durch einen neuen Wert.

Permanente Funktionen

Die permanenten Makrofunktionen unterstützen die Transaktionsverarbeitung in Net.Data, indem Sie definieren können, welche Makroblöcke in einer einzelnen Transaktion permanent sind. Verwenden Sie diese Funktionen, um den Anfang und das Ende einer Transaktion zu definieren und den Geltungsbereich der Variablen in der Transaktion zu definieren sowie um festzulegen, welche HTML-Blöcke in der Transaktion permanent sind und ob für Änderungen in der Transaktion eine COMMIT- oder eine ROLLBACK-Operation ausgeführt werden soll.

Zum Beispiel gibt DTW_ACCEPT die Transaktionskennung für eine Transaktion an, während DTW_TERMINATE den letzten HTML-Block in der Transaktion angibt. DTW_RTVHANDLE generiert eine eindeutige Transaktionskennung für Blöcke in der Transaktion. Sie können DTW_COMMIT und DTW_ROLLBACK verwenden, um in der Transaktion COMMIT- und ROLLBACK-Operationen einzuleiten.

Weitere Informationen hierzu finden Sie in „Transaktionsverwaltung mit permanenten Makros“ auf Seite 123. Eine Liste gültiger permanenter Funktionen mit Syntax und Beispielen finden Sie im Kapitel über integrierte Funktionen im Handbuch *Net.Data Reference*.

Generieren von Web-Seiten in einem Makro

Mit Net.Data können Sie leicht Standard-Web-Seiten im Browser des Anwendungsbenutzers darstellen. In den folgenden Abschnitten werden die HTML- und REPORT-Blöcke des Makros beschrieben und die Formatierung von Web-Seiten in Net.Data-Makros erläutert. Informationen zur Syntax dieser Blöcke finden Sie im Kapitel über die Sprachkonstrukte im Handbuch *Net.Data Reference*.

HTML-Blöcke

Ein Net.Data-Makro enthält HTML-Blöcke, die Textdarstellungsanweisungen wie HTML für einen Web-Browser generieren. In einem Makro muß mindestens ein HTML-Block definiert werden. Jeder HTML-Block generiert eine einzelne Web-Seite beim Browser. Net.Data verarbeitet bei jedem Aufruf nur jeweils einen HTML-Block. Wenn Sie eine Anwendung erstellen möchten, die mehrere Web-Seiten umfaßt, können Sie Net.Data mehrfach aufrufen, um HTML-Blöcke mit normalen Navigationsverfahren wie Programmverbindungen (Links) und Formularen zu verarbeiten.

Alle gültigen Textdarstellungsanweisungen wie HTML oder JavaScript können in einem HTML-Block vorkommen. Darüber hinaus können Sie in einem HTML-Block INCLUDE-Anweisungen, Funktionsaufrufe und Variablenverweise verwenden. Das folgende Beispiel zeigt eine gängige Verwendung von HTML-Blöcken in einem Net.Data-Makro:

```

%DEFINE DATABASE="MNS96"

%HTML(INPUT){
<H1>Hardware Query Form</H1>
<HR>
<FORM METHOD="POST" ACTION="/cgi-bin/db2www/equiplst.d2w/report">
<dl>
<dt>What hardware do you want to list?
<dd><input type="radio" name="hdware" value="MON" checked>Monitors
<dd><input type="radio" name="hdware" value="PNT">Pointing devices
<dd><input type="radio" name="hdware" value="PRT">Printers
<dd><input type="radio" name="hdware" value="SCN">Scanners
</dl>
<HR>
<input type="submit" value="Submit">
</FORM>
%}

%FUNCTION(DTW_SQL) myQuery(){
SELECT MODNO, COST, DESCRIP FROM EQPTABLE WHERE TYPE=$(hdware)
%REPORT{
<B>Here is the list you requested:</B><BR>
%ROW{
<HR>
$(N1): $(V1)      $(N2): $(V2)
<P>
$(V3)
%}
%}
%}

%HTML(REPORT){
@myQuery()
%}

```

Sie können das Net.Data-Makro über eine HTML-Verbindung wie die im folgenden Beispiel aufrufen:

```

<a href="http://www.ibm.com/cgi-bin/db2www/equiplst.d2w/input">
  List of hardware</a>

```

Wenn der Anwendungsbenutzer diese Programmverbindung (Link) anklickt, ruft der Web-Browser Net.Data auf, und Net.Data analysiert das Makro. Wenn Net.Data mit der Verarbeitung des im Aufruf angegebenen HTML-Blocks, in diesem Fall der Block HTML(INPUT), beginnt, fängt Net.Data an, den Text innerhalb des Blocks zu verarbeiten. Alles, was Net.Data nicht als Net.Data-Makrosprachkonstrukt erkennt, wird zum Anzeigen an den Browser gesendet.

Wenn der Benutzer eine Auswahl getroffen und den Knopf zur Übergabe (Submit) angeklickt hat, führt Net.Data den ACTION-Abschnitt des HTML-Elements FORM aus, der einen Aufruf an den Block HTML(OUTPUT) des Net.Data-Makros angibt. Net.Data verarbeitet daraufhin den Block HTML(OUTPUT) ebenso wie zuvor den Block HTML(INPUT).

Anschließend verarbeitet Net.Data den Funktionsaufruf `myQuery()`, der wiederum den SQL-Block `FUNCTION` aufruft. Nachdem der Variablenverweis `$(hardware)` in der SQL-Anweisung durch den vom Eingabeformular zurückgegebenen Wert ersetzt wurde, führt Net.Data die Abfrage aus. An dieser Stelle nimmt Net.Data die Verarbeitung des Berichts wieder auf, der die Ergebnisse der Abfrage gemäß den im `REPORT`-Block definierten Textdarstellungsanweisungen anzeigt.

Nachdem Net.Data die Verarbeitung des `REPORT`-Blocks abgeschlossen hat, kehrt es zum Block `HTML(OUTPUT)` zurück und beendet die Verarbeitung.

REPORT-Blöcke

Das Sprachkonstrukt des `REPORT`-Blocks dient zum Formatieren und Anzeigen der Datenausgabe eines `FUNCTION`-Blocks. Diese Ausgabe besteht in der Regel aus Tabellendaten, obwohl jede gültige Kombination aus Text, Makrovariablenverweisen und Funktionsaufrufen angegeben werden kann.

Wahlfrei kann im `REPORT`-Block ein Tabellenname angegeben werden. Wenn kein Tabellenname angegeben wird, verwendet Net.Data die Tabellendaten aus der ersten Ausgabetabelle in der `FUNCTION`-Parameterliste.

Der `REPORT`-Block besteht aus drei Teilen, die jeweils wahlfrei sind:

- Kopfdaten mit Text, der einmal vor den Daten in der Tabellenzeile angezeigt wird
- Ein `ROW`-Block mit Text und Tabellenvariablen, die einmal für jede Zeile der Ergebnistabelle angezeigt werden
- Fußzeilendaten mit Text, der einmal nach den Daten in der Tabellenzeile angezeigt wird

Beispiel:

```
<H2>Query Results</H2>
<P>Select a name for details.
<TABLE BORDER=1>
<TR>
  <TD>Name</TD>
  <TD>Location</TD></TR>
  %ROW{
<TR>
<TD>
  <a href="/cgi-bin/db2www/name.d2w/details?name=$(V1)&location;=$(V2)">$(V1)</a>
  </TD>
<TD>$(V2)</TD>
  </TR>
  %}
</TABLE>
%}
```

Richtlinien für REPORT-Blöcke

Halten Sie die folgenden Richtlinien ein, wenn Sie REPORT-Blöcke erstellen:

- Wenn Sie keine Tabellenausgabe des ROW-Blocks anzeigen wollen, nehmen Sie keine Angaben im ROW-Block vor, oder übergehen Sie ihn ganz.
- Verwenden Sie verschiedene von Net.Data bereitgestellte Variablen im REPORT-Block für den Zugriff auf die Daten in der Net.Data-Makroergebnistabelle. Diese Variablen sind in „Variablen zur Tabellenverarbeitung“ auf Seite 63 beschrieben. Weitere Einzelheiten finden Sie im Abschnitt über die Berichtsvariablen im Handbuch *Net.Data Reference*.
- Wenn Sie Kopf- und Fußzeilendaten zur Verfügung stellen wollen, geben Sie den Text vor und nach dem ROW-Block an. Net.Data verarbeitet alle Angaben, die vor einem ROW-Block angetroffen werden, als Kopfdaten. Net.Data verarbeitet alle Angaben, die nach dem ROW-Block angetroffen werden, als Fußzeilendaten. Wie beim HTML-Block behandelt Net.Data alle Angaben im Kopfdatenblock, ROW-Block und Fußzeilendatenblock, die nicht als Makrosprachkonstrukte erkannt werden, als Textdarstellungsanweisungen und sendet diese Anweisungen an den Browser.
- Sie können Funktionen und Verweisvariablen in einem REPORT-Block aufrufen.
- Wenn Net.Data einen Standardbericht mit vorformatiertem Text ausgeben soll, nehmen Sie den REPORT-Block nicht in das Makro auf. Das Standardberichtsformat sieht folgendermaßen aus:

```
SHIPDATE | RECDATE | SHIPNO |  
-----  
25/05/1997 | 30/05/1997 | 1495194B |  
-----  
25/05/1997 | 28/05/1997 | 2942821G |  
-----
```

- Wenn Sie die HTML-Befehle anstelle des vorformatierten Textes verwenden wollen, setzen Sie DTW_HTML_TABLE auf YES.
- Um die Ausgabe des Standardberichts zu inaktivieren, setzen Sie DTW_DEFAULT_REPORT auf den Wert NO oder geben einen leeren REPORT-Block an. Beispiel:

```
%REPORT{%}
```

Beispiel: Anpassen eines Berichts

Das folgende Beispiel zeigt, wie Sie Berichtsformate mit Hilfe spezieller Variablen und HTML-Befehle anpassen können. Es werden die Namen, Telefonnummern und Faxnummern aus der Tabelle CustomerTbl angezeigt:

```
%DEFINE SET_TOTAL_ROWS="YES"
...
%FUNCTION(DTW_SQL) custlist() {
    SELECT Name, Phone, Fax FROM CustomerTbl
    %REPORT{
<I>Phone Query Results:</I>
<BR>
=====
<BR>
    %ROW{
        Name: <B>$(V1)</B>
    <BR>
        Phone: $(V2)
    <BR>
        Fax: $(V3)
    <BR>
    -----
    <BR>
        %}
        Total records retrieved: $(TOTAL_ROWS)
    %}
    %}
```

Der hieraus erstellte Bericht sieht im Web-Browser wie folgt aus:

```
Phone Query Results:
=====
Name: Doen, David
Phone: 422-245-1293
Fax: 422-245-7383
-----
Name: Ramirez, Paolo
Phone: 955-768-3489
Fax: 955-768-3974
-----
Name: Wu, Jianli
Phone: 525-472-1234
Fax: 525-472-1234
-----
Total records retrieved: 3
```

Der Bericht wurde von Net.Data wie folgt generiert:

1. Der Titel *Phone Query Results:* wird einmal am Anfang des Berichts ausgegeben. Dieser Text bildet zusammen mit einer Trennlinie den Kopfzeilenbereich des REPORT-Blocks.
2. Die Variablen V1, V2 und V3 werden beim Abruf der einzelnen Zeilen durch die Werte für *Name*, *Phone* und *Fax* ersetzt.
3. Die Zeichenfolge *Total records retrieved:* und der Wert für TOTAL_ROWS werden einmal am Ende des Berichts ausgegeben. (Dieser Text bildet den Fußzeilenbereich des REPORT-Blocks.)

Mehrere REPORT-Blöcke

Sie können mehrere REPORT-Blöcke in einem einzelnen FUNCTION- oder MACRO FUNCTION-Block angeben, um mehrere Berichte mit einem Funktionsaufruf zu generieren.

In der Regel werden mehrere REPORT-Blöcke in der Sprachumgebung DTW_SQL mit einer Funktion verwendet, die eine gespeicherte Prozedur aufruft, die mehrere Ergebnismengen zurückgibt (siehe „Gespeicherte Prozeduren“ auf Seite 112). Mehrere REPORT-Blöcke können jedoch in jeder Sprachumgebung verwendet werden, um mehrere Berichte zu erstellen.

Zur Verwendung mehrerer REPORT-Blöcke müssen Sie eine Net.Data-Tabellenvariable in die Funktionsparameterliste stellen. Wenn mehr Ergebnismengen von der gespeicherten Prozedur zurückgegeben werden als Sie REPORT-Blöcke angegeben haben und wenn die Angabe der integrierten Net.Data-Funktion DTW_DEFAULT_REPORT = "MULTIPLE" ist, werden Standardberichte für jede Tabelle definiert, die keinem REPORT-Block zugeordnet ist. Wenn keine REPORT-Blöcke angegeben sind und DTW_DEFAULT_REPORT = "YES" ist, wird nur ein Standardbericht generiert. Beachten Sie, daß der Wert "YES" für DTW_DEFAULT_REPORT dem Wert "MULTIPLE" nur in der SQL-Sprachumgebung entspricht.

Beispiele: Die folgenden Beispiele zeigen, wie Sie mehrere REPORT-Blöcke verwenden können.

Gehen Sie wie folgt vor, um mehrere Berichte mit den Standardberichtsformaten anzuzeigen:

Beispiel 1: Sprachumgebung DTW_SQL

```
%DEFINE DTW_DEFAULT_REPORT = "MULTIPLE"
%FUNCTION (dtw_sql) myStoredProc (OUT table1, table2) {
    CALL myproc %}
```

In diesem Beispiel gibt die gespeicherte Prozedur myproc zwei Ergebnismengen zurück, die in table1 und table2 gestellt werden. Weil keine REPORT-Blöcke angegeben sind, werden Standardberichte für beide Tabellen angezeigt, zuerst für table1 und dann für table2.

Beispiel 2: MACRO_FUNCTION-Block. In diesem Beispiel werden zwei Tabellen an den MACRO_FUNCTION-Block übergeben. Wenn DTW_DEFAULT_REPORT="MULTIPLE" angegeben ist, werden für beide Tabellen Standardberichte generiert.

```
%DEFINE DTW_DEFAULT_REPORT = "MULTIPLE"
%MACRO_FUNCTION multReport (INOUT tablename1, tablename2) {
    %}
```

In diesem Beispiel werden zwei Tabellen an die MACRO_FUNCTION-Funktion multReport übergeben. Net.Data zeigt wiederum Standardberichte für die zwei Tabellen in der Reihenfolge an, in der sie in der Parameterliste des MACRO FUNCTION-Blocks erscheinen: zuerst für table1 und dann für table2.

Beispiel 3: Sprachumgebung DTW_REXX

```
%DEFINE DTW_DEFAULT_REPORT = "YES"
%FUNCTION (dtw_rexx) multReport (INOUT table1, table2) {
    SAY 'Generating multiple default reports...<BR>'
%}
```

In diesem Beispiel werden zwei Tabellen an die REXX-Funktion `multReport` übergeben. Da `DTW_DEFAULT_REPORT="YES"` angegeben ist, zeigt `Net.Data` nur für die erste Tabelle einen Standardbericht an.

Gehen Sie wie folgt vor, um mehrere Berichte durch Angabe von REPORT-Blöcken zur Anzeigeverarbeitung anzuzeigen:

Beispiel 1: Benannte REPORT-Blöcke

```
%FUNCTION(dtw_sql) myStoredProc (OUT table1, table2) {
    CALL myproc (table1, table2)

    %REPORT(table2) {
        ...
        %ROW { .... }
        ...
    }

    %REPORT(table1) {
        ...
    %row { .... }
        ...
    }
%}
```

In diesem Beispiel wurden REPORT-Blöcke für beide der in der Parameterliste des FUNCTION-Blocks angegebenen Tabellen angegeben. Die Tabellen werden in der Reihenfolge angezeigt, in der sie in den REPORT-Blöcken angegeben sind: `table2` zuerst und dann `table1`. Durch Angabe eines Tabellennamens im REPORT-Block können Sie die Reihenfolge steuern, in der die Berichte angezeigt werden.

Beispiel 2: Nicht benannte REPORT-Blöcke

```
%FUNCTION(dtw_sql) myStoredProc (OUT table1, table2) {  
    CALL myproc  
  
    %REPORT {  
        ...  
        %ROW { .... %}  
        ...  
    %}  
    %REPORT {  
        ...  
        %ROW { .... %}  
        ...  
    %}  
%}
```

In diesem Beispiel wurden REPORT-Blöcke für beide der in der Parameterliste des FUNCTION-Blocks angegebenen Tabellen angegeben. Da keine Tabellennamen in den REPORT-Blöcken angegeben sind, werden Berichte für die zwei Tabellen in der Reihenfolge angezeigt, in der sie von der gespeicherten Prozedur zurückgegeben werden.

Gehen Sie wie folgt vor, um mehrere Berichte mit einer Kombination aus Standardberichten und REPORT-Blöcken anzuzeigen:

Beispiel: Eine Kombination aus Standardberichten und REPORT-Blöcken

```
%DEFINE DTW_DEFAULT_REPORT = "MULTIPLE"  
%FUNCTION(dtw_system) editTables (INOUT table1, table2, table3) {  
    %EXEC{ /qsys.lib/mylib.lib/mypgm.pgm %}  
    %REPORT(table2) {  
        ...  
        %ROW { .... %}  
        ...  
    %}  
%}
```

In diesem Beispiel ist nur ein REPORT-Block angegeben. Weil er den Tabellennamen table2 angibt, wird diese Tabelle verwendet, um den Bericht anzuzeigen. Da weniger REPORT-Blöcke angegeben sind als Ergebnismengen von der gespeicherten Prozedur übergeben werden, werden Standardberichte für die restlichen Ergebnismengen in der folgenden Reihenfolge angezeigt: zuerst ein Standardbericht für table1 und dann ein Standardbericht für table3.

Richtlinien und Einschränkungen für mehrere REPORT-Blöcke: Beachten Sie die folgenden Richtlinien und Einschränkungen für die Angabe mehrerer REPORT-Blöcke in einem FUNCTION- oder MACRO_FUNCTION-Block.

Richtlinien:

- Sie können einen oder mehrere REPORT-Blöcke pro Ergebnismenge bzw. Tabellennamen angeben. Der für den REPORT-Block angegebene Name muß mit einem entsprechenden Parameter für den Ergebnismengennamen oder Tabellennamen in der Parameterliste des FUNCTION-Blocks übereinstimmen.
- Geben Sie die REPORT-Blöcke für mehrere Tabellen in der Reihenfolge an, in der sie verarbeitet werden sollen.
- Soll die Standardverarbeitung erfolgen, wenn kein REPORT-Block für eine Tabelle angegeben ist, definieren Sie DTW_DEFAULT_REPORT = "MULTIPLE". Beim Aufbau der Web-Seite zeigt Net.Data die Standardberichte für Tabellen nach den Berichten für Tabellen mit REPORT-Blöcken an. Beachten Sie, daß die Einstellung DTW_DEFAULT_REPORT = "YES" zur Generierung eines Standardberichts für nur eine Tabelle führt, wenn kein REPORT-Block angegeben wurde. Eine Ausnahme stellt die SQL-Sprachumgebung dar, bei der der Wert YES zur gleichen Verarbeitung führt wie der Wert MULTIPLE.
- Wenn Net.Data Tabellen ohne REPORT-Blöcke nicht anzeigen soll, definieren Sie DTW_DEFAULT_REPORT = "NO".
- Bei Verwendung der Variablen DTW_SAVE_TABLE_IN mit einer Funktion, die mehrere Tabellen zurückgibt, wird die erste von der Funktion zurückgegebene Tabelle der Tabelle DTW_SAVE_TABLE_IN zugeordnet.
- Mehrere REPORT-Blöcke können in jeder Sprachumgebung verwendet werden.

Einschränkungen:

- Die Werte aller Berichtsvariablen in einer Funktion gelten für alle REPORT-Blöcke in dieser Funktion. Sie können den Wert einer Berichtsvariablen nicht für einzelne REPORT-Blöcke ändern.
- Ein MESSAGE-Block muß sich entweder vor oder nach einer Reihe von REPORT-Blöcken befinden. Er darf nicht zwischen REPORT-Blöcken stehen.
- Tabellenvariablen müssen mit der Anweisung TABLE definiert werden, bevor sie an eine Funktion übergeben werden können.
- Wenn der erste REPORT-Block einen Tabellennamen angibt, dann müssen alle REPORT-Blöcke Tabellennamen angeben.
- Wenn der erste REPORT-Block keinen Tabellennamen angibt, dann können keine REPORT-Blöcke Tabellennamen angeben.

Bedingte Logik und Schleifen in einem Makro

Net.Data ermöglicht Ihnen, bedingte Logik und Schleifen in Ihr Net.Data-Makro mit Hilfe von IF- und WHILE-Blöcken zu integrieren.

IF- und WHILE-Blöcke verwenden eine Bedingungsliste, mit der Sie eine oder mehrere Bedingungen testen können. Anschließend können Sie auf der Grundlage des Ergebnisses des Bedingungstests einen Block von Anweisungen ausführen. Die Bedingungsliste enthält logische Operatoren, wie = und <+, und Terme, die sich aus Zeichenfolgen in Anführungszeichen, Variablen, Variablenverweisen und Funktionsaufrufen zusammensetzen. Zeichenfolgen in Anführungszeichen können zudem Variablenverweise und Funktionsaufrufe enthalten. Sie können die Bedingungsliste verschachteln.

In den folgenden Abschnitten wird die Verwendung der bedingten Logik und von Schleifen beschrieben:

- „Bedingte Logik: IF-Blöcke“
- „Schleifenkonstrukte: WHILE-Blöcke“ auf Seite 90

Bedingte Logik: IF-Blöcke

Mit Hilfe des IF-Blocks können Sie in einem Net.Data-Makro eine bedingte Verarbeitung durchführen. Der IF-Block ist den IF-Anweisungen der meisten höheren Programmiersprachen ähnlich, weil er die Möglichkeit bietet, eine oder mehrere Bedingungen zu testen und anschließend auf der Grundlage des Ergebnisses des Bedingungstests einen Block von Anweisungen auszuführen.

Sie können IF-Blöcke fast überall in einem Makro verwenden und verschachteln. Die Syntax eines IF-Blocks wird im Kapitel über die Sprachkonstrukte im Handbuch *Net.Data Reference* dargestellt.

Regeln für IF-Block: Die Syntaxregeln für einen IF-Block werden durch die Position des Blocks im Makro bestimmt. Die Elemente, die im Block der ausführbaren Anweisungen eines IF-Blocks zulässig sind, hängen von der Position des IF-Blocks selbst ab.

- Jedes Element, das innerhalb des Blocks, in dem sich der IF-Block befindet, gültig ist, ist auch innerhalb dieses IF-Blocks gültig. Wenn Sie zum Beispiel einen IF-Block innerhalb eines HTML-Blocks angeben, ist jedes Element, das in dem HTML-Block zulässig ist, auch in diesem IF-Block zulässig, wie zum Beispiel INCLUDE-Anweisungen und WHILE-Blöcke.

```
%HTML block
...
    %IF block
...
    %INCLUDE
...
    %WHILE
...
%ENDIF
%}
```

- Wenn Sie den IF-Block außerhalb jedes anderen Blocks im Deklarationsabschnitt des Net.Data-Makros angeben, sind analog nur solche Elemente in dem IF-Block zulässig, die außerhalb jedes anderen Blocks zulässig sind (wie zum Beispiel ein DEFINE-Block oder ein FUNCTION-Block).

```
%IF
...
    %DEFINE
...
    %FUNCTION
...
%ENDIF
```

- Wenn ein IF-Block in einem IF-Block verschachtelt ist, der sich außerhalb jedes anderen Blocks im Deklarationsabschnitt befindet, können in diesem Block alle Elemente verwendet werden, die im außerhalb liegenden Block verwendet werden können. Wenn ein IF-Block in einem anderen Block verschachtelt ist, der sich in einem IF-Block befindet, übernimmt er die Syntaxregeln des Blocks, in dem er sich befindet.

Im folgenden Beispiel muß der verschachtelte IF-Block den Regeln folgen, die innerhalb eines HTML-Blocks gelten.

```
%IF
...
    %HTML {
...
    %IF
...
    %ENDIF
    %}
...
%ENDIF
```

Ausnahme: Geben Sie keinen ROW-Block in einem IF-Block an.

Zeichenfolgevergleiche für IF-Blöcke

Net.Data verarbeitet die Bedingungsliste des IF-Blocks auf eine von zwei Arten, je nach dem Inhalt der Ausdrücke, aus denen die Bedingungen bestehen. Die Standardaktion besteht darin, alle Ausdrücke als Zeichenfolgen zu behandeln und Zeichenfolgevergleiche wie in den Bedingungen angegeben durchzuführen. Wenn jedoch zwei Zeichenfolgen, die ganze Zahlen darstellen, verglichen werden, ist der Vergleich numerisch. Net.Data geht davon aus, daß eine Zeichenfolge numerisch ist, wenn sie nur Ziffern enthält, denen wahlfrei ein Zeichen '+' oder '-' vorangestellt sein kann. Die Zeichenfolge darf keine Nichtziffernzeichen außer '+' bzw. '-' enthalten. Net.Data unterstützt keinen numerischen Vergleich von nicht ganzzahligen Zahlen.

Beispiele für gültige Ganzzahlenfolgen:

```
+1234567890  
-47  
000812  
92000
```

Beispiele für ungültige Ganzzahlenfolgen:

```
- 20      (enthält Leerzeichen)  
234,000   (enthält ein Komma)  
57.987    (enthält einen Punkt)
```

Net.Data wertet die IF-Bedingung zum Zeitpunkt der Ausführung des Blocks aus. Dieser Zeitpunkt muß nicht mit dem Zeitpunkt, zu dem der Block ursprünglich von Net.Data gelesen wurde, übereinstimmen. Wenn Sie zum Beispiel einen IF-Block in einem REPORT-Block angeben, wertet Net.Data die Bedingungsliste des IF-Blocks nicht beim Lesen der FUNCTION-Blockdefinition mit dem REPORT-Block aus, sondern erst beim Aufrufen und Ausführen der Funktion. Dies gilt sowohl für den Abschnitt mit der Bedingungsliste des IF-Blocks als auch für den Block der auszuführenden Anweisungen.

Beispiel für IF-Block: Ein Makro mit IF-Blöcken in anderen Blöcken

```
%{ This macro is called from another macro, passing the operating system
    and version variables in the form data.
%}
```

```
%IF (platform == "AS400")
  %IF (version == "V3R2")
    %INCLUDE "as400v3r2_def.hti"
  %ELIF (version == "V3R7")
    %INCLUDE "as400v3r7_def.hti"
  %ELIF (version == "V4R1")
    %INCLUDE "as400v4r1_def.hti"
%ENDIF
%ELSE
  %INCLUDE "default_def.hti"
%ENDIF
```

```
%MACRO_FUNCTION numericCompare(IN term1, term2, OUT result) {
%IF (term1 < term2)
  @dtw_assign(result, "-1")
%ELIF (term1 > term2)
  @dtw_assign(result, "1")
%ELSE
  @dtw_assign(result, "0")
%ENDIF
%}
```

```
%HTML(report){
  %WHILE (a < "10") {
    outer while loop #$(a)<BR>
    %IF (@dtw_rdivrem(a,"2") == "0")
      this is an even number loop<BR>
    %ENDIF
    @DTW_ADD(a, "1", a)
  %}
%}
```

Schleifenkonstrukte: WHILE-Blöcke

Mit Hilfe des WHILE-Blocks können Schleifen in einem Net.Data-Makro durchgeführt werden. Wie der IF-Block bietet der WHILE-Block die Möglichkeit, eine oder mehrere Bedingungen zu testen und anschließend auf der Grundlage des Ergebnisses des Bedingungstests einen Block von Anweisungen auszuführen. Im Gegensatz zum IF-Block kann der Anweisungsblock auf der Grundlage des Ergebnisses des Bedingungstests beliebig oft ausgeführt werden.

Sie können WHILE-Blöcke innerhalb von HTML-Blöcken, REPORT-Blöcken, ROW-Blöcken, MACRO_FUNCTION-Blöcken und IF-Blöcken angeben, und Sie können sie verschachteln. Die Syntax eines WHILE-Blocks wird im Kapitel über die Sprachkonstrukte im Handbuch *Net.Data Reference* gezeigt.

Net.Data verarbeitet den WHILE-Block genau so wie den IF-Block, jedoch wird die Bedingung nach jeder Ausführung des Blocks erneut ausgewertet. Und wie bei jedem bedingten Schleifenkonstrukt kann die Verarbeitung zu einer Endlosschleife führen, wenn die Bedingung nicht korrekt codiert wurde.

Beispiel: Ein Makro mit einem WHILE-Block

```
%DEFINE loopCounter = "1"

%HTML(build_table) {
%WHILE (loopCounter <= "100") {
  %{ generate table tag and column headings %}
  %IF (loopCounter == "1")
    <TABLE BORDER>
    <TR>
    <TH>Item #
    <TH>Description
  %ENDIF

  %{ generate individual rows %}
  <TR>
  <TD>$(loopCounter)
  <TD>@getDescription(loopCounter)

  %{ generate end table tag %}
  %IF (loopCounter == "100")
%ENDIF

  %{ increment loop counter %}
  @DTW_ADD(loopCounter, "1", loopCounter)
  %}
%}
```

Verwenden der Sprachumgebungen

Net.Data stellt Sprachumgebungen bereit, mit denen Sie auf Datenquellen zugreifen und Anwendungsprogramme mit Geschäftslogik ausführen können. Mit der SQL-Sprachumgebung können Sie zum Beispiel SQL-Anweisungen an eine DB2-Datenbank übergeben, und mit der REXX-Sprachumgebung können Sie REXX-Programme aufrufen. Mit der SYSTEM-Sprachumgebung können Sie ein Programm ausführen oder einen Befehl absetzen.

In Net.Data können Sie benutzerdefinierte Sprachumgebungen wie Plug-Ins hinzufügen. Jede benutzerdefinierte Sprachumgebung muß eine Standardgruppe von Schnittstellen unterstützen, die von Net.Data definiert werden, und muß als Serviceprogramm implementiert werden. Ausführliche Informationen zum Erstellen einer benutzerdefinierten Sprachumgebung finden Sie im Handbuch *Net.Data Language Environment Interface Reference*.

Abb. 8 zeigt die Beziehung zwischen dem Web-Server, Net.Data und den Net.Data-Sprachumgebungen.

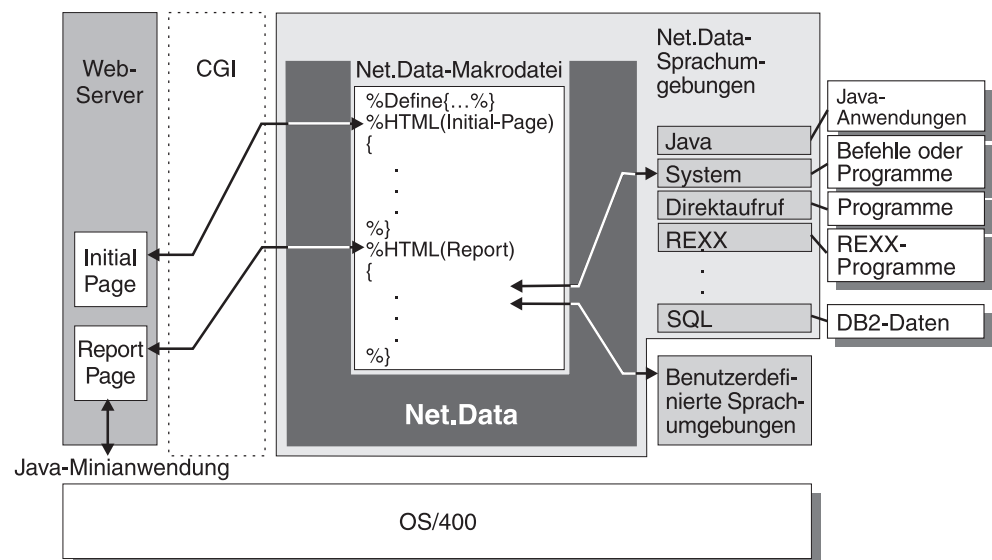


Abbildung 8. Die Net.Data-Sprachumgebungen

In den folgenden Abschnitten werden die Net.Data-Sprachumgebungen und ihre Verwendung in Ihren Makros beschrieben:

- „Übersicht über die von Net.Data bereitgestellten Sprachumgebungen“ auf Seite 92
- „Aufrufen einer Sprachumgebung“ auf Seite 92
- „Direktaufrufsprachumgebung“ auf Seite 93
- „Java-Anwendungssprachumgebung“ auf Seite 98
- „REXX-Sprachumgebung“ auf Seite 99
- „SQL-Sprachumgebung“ auf Seite 102
- „SYSTEM-Sprachumgebung“ auf Seite 120

Informationen zur Leistungssteigerung bei Verwendung der Sprachumgebungen finden Sie in „Optimieren der Sprachumgebungen“ auf Seite 135.

Übersicht über die von Net.Data bereitgestellten Sprachumgebungen

Net.Data stellt Sprachumgebungen bereit, mit denen Sie auf Daten und Programmierungsressourcen für Ihre Anwendung zugreifen können.

Tabelle 3 liefert eine Kurzbeschreibung jeder Sprachumgebung.

Tabelle 3. Net.Data-Sprachumgebungen

Sprachumgebung	Umgebungsname	Beschreibung
Direktaufruf	DTW_DIRECTCALL	Die Direktaufrufsprachumgebung unterstützt Aufrufe an externe Programme, die in einer höheren Programmiersprache wie RPG, COBOL oder C/C++ geschrieben wurden.
Java-Anwendung	DTW_JAVAPPS	Net.Data unterstützt Ihre vorhandenen Java-Anwendungen in der Java-Sprachumgebung.
REXX	DTW_REXX	Die REXX-Sprachumgebung interpretiert interne REXX-Programme, die in einem FUNCTION-Block des Net.Data-Makros angegeben sind, oder sie führt externe REXX-Programme aus, die in einer separaten Datei gespeichert sind.
SQL	DTW_SQL	Die SQL-Sprachumgebung führt SQL-Anweisungen über DB2 aus. Die Ergebnisse der SQL-Anweisung können in einer Tabellenvariablen zurückgegeben werden.
System	DTW_SYSTEM	Die SYSTEM-Sprachumgebung unterstützt das Ausführen von Befehlen und das Aufrufen externer Programme.

Aufrufen einer Sprachumgebung

Gehen Sie wie folgt vor, um eine Sprachumgebung aufzurufen:

- Definieren Sie mit einer Anweisung FUNCTION eine Funktion, die die Sprachumgebung aufruft.
- Verwenden Sie einen Funktionsaufruf an die Sprachumgebung.

Beispiel:

```
%FUNCTION(DTW_SQL) custinfo() {  
  select customer, custno from customer.data  
  %}  
...  
%HTML(REPORT){  
  @custinfo()  
  %}
```


Behandeln von Fehlerbedingungen

Wenn in einer Sprachumgebungsfunktion ein Fehler festgestellt wird, legt die Sprachumgebung die Net.Data-Variable RETURN_CODE mit einem Fehlercode fest.

Sie können Fehlerbedingungen mit Hilfe der folgenden Ressourcen behandeln:

- Die von Net.Data bereitgestellten Sprachumgebungen geben Fehlercodes zurück, die im Handbuch *Net.Data Messages and Codes Reference* dokumentiert sind.
- Die Datenbanksprachumgebungen wie die SQL-Sprachumgebung weisen der Variablen RETURN_CODE Fehlercodes (sogenannte SQLCODE-Werte) zu, die durch das Datenbankverwaltungssystem (DBMS) zurückgegeben werden. Weitere Informationen zu den von Ihrem DBMS verwendeten SQLCODE-Werten finden Sie im Handbuch zu Fehlermeldungen Ihres DBMSs.

Sicherheit

Stellen Sie sicher, daß die Benutzer-ID, unter der Net.Data ausgeführt wird, über die entsprechende Berechtigung zum Zugreifen auf ein Objekt verfügt, auf das eventuell vom Ziel einer Sprachumgebungsanweisung verwiesen wird. Die SQL-Sprachumgebung z. B. führt SQL-Anweisungen aus, und SQL-Anweisungen greifen auf Datenbankdateien zu. Das heißt, daß die Benutzer-ID, unter der Net.Data ausgeführt wird, über die Berechtigung für die Datenbankdateien verfügen muß.

Direktaufrufsprachumgebung

Die Direktaufrufsprachumgebung ermöglicht den Aufruf von Programmen, die in einer höheren Programmiersprache wie C, RPG, COBOL oder CL geschrieben wurden. Parameter können *an* das Programm übergeben werden, und Parameterwerte können *vom* Programm empfangen werden. Dies ermöglicht einfache Integration vorhandener Programme in Net.Data und befähigt Benutzer, vorhandene Programmiererfahrung zum Codieren komplizierter Geschäftslogik einzusetzen.

Aufrufen von Programmen

Definieren Sie zum Aufrufen eines Programms eine Funktion, die die Direktaufrufsprachumgebung (DTW_DIRECTCALL) verwendet und einen Pfad zu dem Programm enthält, das in einer EXEC-Anweisung aufgerufen werden soll. Beispiel:

```
%function(DTW_DIRECTCALL) dc1() {  
    %EXEC { /QSYS.LIB/NETDATA.LIB/MYPPGM.PGM %}  
%}
```

Sie können den Pfad zum Programm kürzen, wenn Sie die Konfigurationsvariable EXEC_PATH zum Definieren von Pfaden zu Verzeichnissen mit Programmen verwenden. Informationen zum Definieren der Konfigurationsvariablen EXEC_PATH finden Sie in „EXEC_PATH“ auf Seite 17.

Unterstützte Sprachumgebungsvariablen

Die Direktaufsprachumgebung unterstützt die Variable DTW_PAD_PGM_PARMS, die angibt, ob an ein Programm zu übergebende Parameter bis zur angegebenen Genauigkeit mit Leerzeichen aufgefüllt werden sollen. Im Handbuch *Net.Data Reference* finden Sie eine Beschreibung, eine Syntaxdarstellung und Beispiele für diese Variable. Weitere Informationen zum Übergeben von Parametern an Programme finden Sie in „Übergeben von Parametern an Programme“.

Übergeben von Parametern an Programme

Übergeben Sie Parameter an ein Programm, indem Sie in der Funktionsdefinition den Datentyp des Parameters angeben und definieren, daß der zu übergebende Parameter nur zur Eingabe (IN), nur zur Ausgabe (OUT) oder zur Eingabe/Ausgabe (INOUT) dienen soll. Beispiel:

```
%function(DTW_DIRECTCALL) dc2(IN CHAR(3) p1,  
                               INOUT INTEGER p2,  
                               OUT DECIMAL(7,2) p3) {  
    %EXEC { /QSYS.LIB/NETDATA.LIB/MYPPGM.PGM %}  
}%
```

Im obigen Beispiel übergibt die Direktaufsprachumgebung drei Parameter, eine Zeichenvariable, eine ganze Zahl und eine gepackte Dezimalvariable an das Programm MYPPGM. Sie können bis zu 50 Parameter an das aufgerufene Programm übergeben. Nur mit Datentypen angegebene Parameter werden an das Programm übergeben. Die Direktaufsprachumgebung setzt die dem Parameter entsprechende Zeichenfolge in die interne Darstellung des Datentyps um. Die Sprachumgebung übergibt dann Zeiger auf die interne Darstellung der Variablen an das aufgerufene Programm, und zwar in der in der Funktionsdefinition angegebenen Reihenfolge.

Da Zeiger auf die Variablen an das Programm übergeben werden, kann das Programm den Wert der Variablen ändern. Es werden jedoch nur die vom Programm geänderten OUT- oder INOUT-Variablen an das Makro zurückgemeldet, das die Sprachumgebung aufgerufen hat.

Unterstützte Datentypen

Tabelle 4 auf Seite 95 listet die von der Direktaufsprachumgebung unterstützten Datentypen auf. Nicht alle Datentypen werden von jeder höheren Programmiersprache unterstützt.

Tabelle 4. Datentypen für Direktaufruf

Datentyp	Hinweise
CHAR(<i>n</i>) CHARACTER(<i>n</i>) CHARACTER	Eine Zeichenfolge. Wenn <i>n</i> angegeben wird, muß die Zeichenfolge größer als Null sein. Wenn die Zeichenfolge nicht angegeben wird, wird davon ausgegangen, daß sie ein Zeichen ist. Da alle von der Direktaufsprachumgebung übergebenen Zeichenfolgen auf Null enden, ordnet die Sprachumgebung <i>n</i> +1 Byte zu (1 Byte für das Nullabschlußzeichen). Zeichenfolgen, die <i>n</i> überschreiten, werden abgeschnitten.
VARCHAR(<i>n</i>)	Eine Zeichenfolge variabler Länge, wobei <i>n</i> größer als Null und kleiner-gleich 32740 ist. Die Zeichenfolge endet auf Null, und die Sprachumgebung ordnet <i>n</i> +2+1 Byte zu (2 Byte zum Speichern der Zeichenfolgelänge, 1 Byte für das Nullabschlußzeichen). Zeichenfolgen, die <i>n</i> überschreiten, werden abgeschnitten. Die ersten beiden Byte der Zeichenfolge enthalten die Zeichenfolgelänge (binärer Wert). Wenn der Parameter als OUT definiert ist (nur Ausgabe), wird die Zeichenfolgelänge auf Null gesetzt, bevor die Variable an das aufgerufene Programm übergeben wird.
INTEGER INT	Eine binäre Ganzzahl mit Vorzeichen, 4 Byte lang
SMALLINT	Eine binäre Ganzzahl mit Vorzeichen, 2 Byte lang
FLOAT(<i>p,s</i>)	Eine Gleitkommazahl einfacher Genauigkeit oder doppelter Genauigkeit. Bei einfacher Genauigkeit muß <i>p</i> größer als 0 und kleiner als 25 sein. Bei doppelter Genauigkeit muß <i>p</i> größer als 24 und kleiner als 54 sein. Die Genauigkeit (<i>p</i>) und die Anzahl der Kommastellen (<i>s</i>) werden nur beim Umsetzen von Daten in ein anzeigbares Format verwendet, z. B. in eine Zeichenfolge.
REAL(<i>p,s</i>)	Eine Gleitkommazahl einfacher Genauigkeit. <i>p</i> muß größer als 0 und kleiner als 25 sein. Die Genauigkeit (<i>p</i>) und die Anzahl der Kommastellen (<i>s</i>) werden nur beim Umsetzen von Daten in ein anzeigbares Format verwendet, z. B. in eine Zeichenfolge.
DOUBLE (<i>p,s</i>) DOUBLEPRECISION(<i>p,s</i>)	Eine Gleitkommazahl doppelter Genauigkeit. <i>p</i> muß größer als 0 und kleiner als 53 sein. Die Genauigkeit (<i>p</i>) und die Anzahl der Kommastellen (<i>s</i>) werden nur beim Umsetzen von Daten in ein anzeigbares Format verwendet, z. B. in eine Zeichenfolge.
NUMERIC(<i>p,s</i>)	Eine gezonte Dezimalzahl mit Genauigkeit (<i>p</i>) und Anzahl der Kommastellen (<i>s</i>). Der Wert von <i>p</i> muß größer als 0 und kleiner als 32 sein.
DEC(<i>p,s</i>) DECIMAL(<i>p,s</i>)	Eine gepackte Dezimalzahl mit Genauigkeit (<i>p</i>) und Anzahl der Kommastellen (<i>s</i>). Der Wert von <i>p</i> muß größer als 0 und kleiner als 32 sein.
DTWTABLE	Ein spezieller Datentyp zum Übergeben einer Net.Data-Tabelle an das aufgerufene Programm. Die Direktaufsprachumgebung übergibt einen Zeiger auf die Tabelle, der mit den Tabellenfunktionen der Net.Data-Sprachumgebungsschnittstelle bearbeitet werden kann.

Numerisch definierte Parameter können das Währungssymbol und dreistellige Trennzeichen enthalten. Die Direktaufsprachumgebung entfernt das Währungssymbol und die dreistelligen Trennzeichen beim Umsetzen einer numerischen Variable vom Zeichenfolgeformat in das interne Format, bevor die Variable an das Programm übergeben wird. Net.Data ruft das Währungssymbol, das Dezimalformat und die dreistelligen Trennzeichen aus den Prozeßattributen des Prozesses ab, in dem Net.Data ausgeführt wird.

Parameter für auf Null endende Zeichenfolgen

Wenn DTW_PAD_PGM_PARMS in der Konfigurationsdatei oder im Makro auf NO gesetzt wird, übergibt die Direktaufsprachumgebung Zeichenfolgewerte mit einem Nullabschlußzeichen (Wert x'00') an Ihr Programm. Daher müssen Sie Code zum Verarbeiten der Zeichenfolge schreiben (außer wenn Sie C oder C++ verwenden, die auf Null endende Zeichenfolgen erwarten).

Wenn Sie z. B. das Parameterfeld als CHAR(10) definieren, aber einen Zeichenfolgewert übergeben, der 5 Byte lang ist, stellt Net.Data das Nullabschlußzeichen hinter das fünfte Byte. Die Übergabe des Werts "12345" als Zeichenfolge in einem Feld mit CHAR(10) ergibt folgendes:

```
x'F1F2F3F4F500.....'
```

Die Byte nach dem Nullabschlußzeichen sind nicht definiert (Sie können nicht davon ausgehen, daß die Byte Null oder Leerzeichen sind).

Da die Zeichenfolge auf Null endet und nicht initialisierte Byte nach dem Nullabschlußzeichen enthält, können Sie die Zeichenfolge nicht in einem RPG- oder COBOL-Programm verwenden. Wenn Sie z. B. die Zeichenfolge in einer Vergleichsoperation verwenden, gibt die Operation keine gültigen Ergebnisse aus. Das Programm erwartet nicht, daß die Zeichenfolge das Nullabschlußzeichen enthält, sondern erwartet vielmehr, daß die Zeichenfolge am Ende mit Leerzeichen aufgefüllt ist. Sie können mit den Zeichenfolgebearbeitungsfunktionen in Ihrem Programm den Zeichenfolgewert extrahieren oder den Datentyp VARCHAR verwenden. Durch diese Methode erhalten Sie die Länge der Zeichenfolge in den ersten beiden Byte.

Wenn DTW_PAD_PGM_PARMS in der Konfigurationsdatei oder im Makro auf YES gesetzt wird, übergibt die Direktaufsprachumgebung Zeichenfolgewerte an Ihr Programm. Dabei sind die Werte nach rechts bis zur Genauigkeitslänge mit Leerzeichen aufgefüllt. Wenn Sie das gleiche Beispiel wie oben verwenden, DTW_PAD_PGM_PARMS jedoch auf YES setzen, ergibt die Übergabe des Werts "12345" als Zeichenfolge in einem Feld mit CHAR(10) folgendes:

```
x'F1F2F3F4F54040404000'
```

Da die Zeichenfolge die Länge 5 hat, die kleiner ist als die angegebene Genauigkeit, werden nach dem Wert bis zur Genauigkeitslänge Leerzeichen eingefügt. In Sprachen wie RPG geschriebene Programme können jetzt den Parameter verwenden, ohne daß sie auf Null endende Zeichenfolgen handhaben müssen.

Allgemeine Fehler beim Übergeben von Parametern

In der folgenden Liste werden Fehler beschrieben, die auftreten können, wenn über die Direktaufsprachumgebung Programme aufgerufen und Parameter an das Programm übergeben werden. Es werden auch Tips zum Vermeiden dieser Fehler aufgeführt.

Parameterabweichungsfehler

Stellen Sie sicher, daß die Anzahl und Reihenfolge der Parameter mit der Anzahl und Reihenfolge übereinstimmen, in der sie in der Parameterliste des aufgerufenen Programms vorkommen.

Datentypfehler

Stellen Sie sicher, daß der für einen Parameter angegebene Datentyp mit dem vom aufgerufenen Programm erwarteten Datentyp übereinstimmt. Von der Direktaufsprachumgebung werden eventuell Datentypen unterstützt, die von der höheren Programmiersprache, mit der das aufgerufene Programm erstellt wurde, nicht unterstützt werden.

Längenfehler

Stellen Sie sicher, daß die für Parameter definierten Längen korrekt sind und mit den im aufgerufenen Programm angegebenen Längen übereinstimmen. Die Angabe einer Länge, die kürzer ist als die deklarierte Länge des aufgerufenen Programms, beschädigt eventuell den Speicher und kann zur Folge haben, daß Net.Data nicht mehr ordnungsmäßig funktioniert.

Zurückgeben von Werten aus Programmen

Einige höhere Programmiersprachen wie C können beim Programmaufruf eine ganze Zahl zurückgeben. Die ganze Zahl kann durch Angabe des Schlüsselworts RETURNS in der Funktionsdefinition abgerufen werden, z. B.:

```
%function(DTW_DIRECTCALL) dc3(IN CHAR(3) p1) RETURNS(retval) {  
    %EXEC { /QSYS.LIB/NETDATA.LIB/MYPPGM.PGM %}  
}%
```

Wenn der Funktionsaufruf erfolgreich abgeschlossen wird, enthält der Parameter *retval* den durch das Programm zurückgegebenen Wert.

Beispiel für die Direktaufsprachumgebung

Im folgenden Beispiel ruft das Makro ein Programm auf und übergibt mehrere Parameter. Nach dem Makro wird die in RPG und CL geschriebene Quelle für das Programm aufgelistet. Das aufgerufene Programm akzeptiert zwei ganzzahlige Parameter. Es kopiert den ersten Parameter (den Eingabeparameter) in den zweiten Parameter (den Ausgabeparameter).

Makro:

```
%define ilepgm = "/QSYS.LIB/NETDATADEV.LIB/TDCCLI01.PGM"  
%define out1 = "0"
```

```
%FUNCTION(DTW_DIRECTCALL) dcFunction(IN INT inp1,  
                                       OUT INT outp2)  
{ %EXEC { $(ilepgm) %} %}
```

```
%HTML(REPORT){  
    @dcFunction("123", out1)  
    The value of out1 is: "${out1}"  
}%
```

ILE-RPG-Programm:

```
DINP1          S          10I00
DOUTP2         S          10I00
C*
C      *ENTRY      PLIST
C          PARM          INP1
C          PARM          OUTP2
C*
C          Z-ADD      INP1      OUTP2
C*
C          SETON          LR
```

CL-Programm:

```
PGM PARM(&INP1; &OUTP2;)

DCL VAR(&INP1;) TYPE(*CHAR) LEN(4)
DCL VAR(&OUTP2;) TYPE(*CHAR) LEN(4)

CHGVAR VAR(&OUTP2;) VALUE(&INP1;)
ENDPGM
```

Java-Anwendungssprachumgebung

Mit der Java-Anwendungssprachumgebung können Sie Java-Programme aufrufen, was die einfache Integration von Java-Anwendungen in Net.Data ermöglicht. Die Java-Anwendungssprachumgebung wurde zuerst in OS/400 Version 4 Release 4 eingeführt. Führen Sie die in „Definieren der Java-Anwendungssprachumgebung“ auf Seite 22 dokumentierten Konfigurationsschritte aus, um mit der Java-Anwendungssprachumgebung zu arbeiten.

Aufrufen von Java-Programmen

Definieren Sie eine Funktion, die die Java-Anwendungssprachumgebung (DTW_JAVAPPS) verwendet, um ein Java-Programm aufzurufen. Geben Sie einen Funktionsnamen für den Klassennamen des Java-Programms an.

Beispiel: Aufrufen des Java-Programms helloWorld.java:

```
%function(DTW_JAVAPPS) helloWorld() { %}
```

Die Java-Anwendungssprachumgebung erwartet, daß Java-Programme eine Methodenkennung für 'main' enthalten, die erste in einem Java-Programm ausgeführte Methode. Wenn die Sprachumgebung eine Anwendung aufruft, hat sie Zugriff auf die Standardeingabe (stdin) und die Standardausgabe (stdout). In der Standardeingabe sind keine Formulardaten vorhanden, weil Net.Data die Daten bereits gelesen hat.

Wichtig: Legen Sie die Konfigurationsvariable DTW_JAVA_CLASSPATH vor dem Aufrufen von Java-Anwendungen fest, damit die Java-Klassen gefunden werden können. Die Syntax dieser Variable wird in „DTW_JAVA_CLASSPATH“ auf Seite 20 angegeben.

Übergeben von Parametern an Java-Programme

Übergeben Sie Parameter an Java-Programme, indem Sie die zu übergebenden Parameter in der Funktionsdefinition angeben. Geben Sie nur Zeichenfolgeparameter an, die nur zur Eingabe (IN) oder zur Ein- und Ausgabe (INOUT) dienen.

Beispiel: Übergeben des IN-Parameters p1 an den Funktionsaufruf

```
%function(DTW_JAVAPPS) jv1(IN p1) { %}
```

Die Java-Anwendungssprachumgebung unterstützt keine Java-Programme, die Variablen aktualisieren, weil die aktualisierten Werte nicht an das Makro zurückgegeben werden können.

Beispiel für die Java-Anwendungssprachumgebung

Im folgenden Beispiel ruft das Net.Data-Makro das Java-Programm `echoString` auf. Das Makro übergibt zwei Zeichenfolgeparameter an die Java-Sprachumgebung. Die erste Zeichenfolge teilt dem Java-Programm mit, kursive oder fette Hervorhebung für den zweiten Parameter, eine Zeichenfolge, zu verwenden, bevor der zweite Parameter an die Standardausgabe (stdout) übergeben wird. Da das Programm "I" für Kursivdarstellung übergibt, zeigt der Web-Server die Zeichenfolge *Hello World* im Browser kursiv an. Nach dem Makro wird die Quelle für das Java-Programm aufgelistet.

Makro:

```
%FUNCTION(DTW_JAVAPPS) echoString(textAttribute, text){ %}
```

```
%HTML(runjava){  
  @echoString("I","Hello World")  
%}
```

Java-Programm:

```
class echoString {  
    public static void main (String args[]) {  
        if (args[0].equals("I"))  
            System.out.println("<I>" + args[1] + "</I>");  
        else  
            System.out.println("<B>" + args[1] + "</B>");  
    }  
}
```

REXX-Sprachumgebung

In der REXX-Sprachumgebung können Sie REXX-Programme ausführen.

Ausführen von REXX-Programmen

In der REXX-Sprachumgebung können Sie sowohl interne REXX-Programme als auch externe REXX-Programme ausführen. Ein internes REXX-Programm ist ein REXX-Programm, dessen Quelle sich im Makro befindet. Bei einem externen REXX-Programm befindet sich die Quelle des REXX-Programms in einer externen Datei.

Gehen Sie wie folgt vor, um ein internes REXX-Programm auszuführen:

Definieren Sie eine Funktion, die die REXX-Sprachumgebung (DTW_REXX) verwendet und den REXX-Code in der Funktion enthält.

Beispiel: Eine Funktion mit einem internen REXX-Programm

```
%function(DTW_REXX) helloWorld() {  
    SAY 'Hello World'  
%}
```

Gehen Sie wie folgt vor, um ein externes REXX-Programm auszuführen:

Definieren Sie eine Funktion, die die REXX-Sprachumgebung (DTW_REXX) verwendet und einen Pfad zu dem REXX-Programm enthält, das in einer EXEC-Anweisung ausgeführt werden soll.

Beispiel: Eine Funktion mit einer EXEC-Anweisung, die auf ein externes Programm zeigt

```
%function(DTW_REXX) externalHelloWorld() {  
%EXEC{ /QSYS.LIB/REXX.LIB/REXXSRC.FILE/HELLOWORLD.MBR%}  
%}
```

Sie können den Pfad zum Programm kürzen, wenn Sie die Konfigurationsvariable EXEC_PATH zum Definieren von Pfaden zu Verzeichnissen mit Programmen verwenden. Informationen zum Definieren der Konfigurationsvariablen EXEC_PATH finden Sie in „EXEC_PATH“ auf Seite 17.

Einschränkung: Wenn Sie OS/400 Version 3 Release 2 oder Version 3 Release 7 ausführen und wenn ein REXX-Programm die REXX-Anweisung SAY verwendet, um in stdout zu schreiben, müssen Sie 12 Leerzeichen am Anfang der Zeichenfolge einfügen. Beispiel:

```
SAY '          STARTOFDATA'
```

Die 12 Leerzeichen werden ignoriert. Werden sie jedoch nicht eingefügt, kann dies zu unvorhersehbaren Ergebnissen führen.

Übergeben von Parametern an REXX-Programme

Es gibt zwei Möglichkeiten, Informationen an ein REXX-Programm zu übergeben, das durch die REXX-Sprachumgebung (DTW_REXX) aufgerufen wird: direkt und indirekt.

Direkt Mit der Anweisung %EXEC übergeben Sie Parameter direkt an ein externes REXX-Programm. Beispiel:

```
%FUNCTION(DTW_REXX) rexx1() {  
    %EXEC{  
        /QSYS.LIB/NETDATA.LIB/QREXXSRC.FILE/CALL1.MBR ${INPARAM1} %}  
    %}
```

Der Verweis auf die Net.Data-Variable INPARAM1 wird aufgehoben, und die Variable wird an das externe REXX-Programm übergeben. Das REXX-Programm kann durch Verwendung der Anweisung REXX PARSE ARG auf die Variable verweisen.

Die Parameter, die dem Programm mit dieser Methode übergeben werden, werden als Eingabeparameter behandelt (die dem Programm übergebenen Parameter können vom Programm verwendet und bearbeitet werden, Änderungen an den Parametern werden jedoch nicht zurück an Net.Data übertragen).

Indirekt

Mit dem *Variablenpool* des REXX-Programms übergeben Sie Parameter indirekt. Wenn ein REXX-Programm gestartet wird, wird vom REXX-Interpreter ein Bereich, der Informationen zu allen Variablen enthält, erstellt und verwaltet. Dieser Bereich wird als Variablenpool bezeichnet.

Wenn eine Funktion der REXX-Sprachumgebung (DTW_REXX) aufgerufen wird, werden Eingabeparameter (IN) oder Eingabe-/Ausgabeparameter (INOUT) von der REXX-Sprachumgebung im Variablenpool gespeichert, bevor das REXX-Programm ausgeführt wird. Wenn das REXX-Programm aufgerufen wird, kann es direkt auf diese Variablen zugreifen. Nach erfolgreicher Beendigung des REXX-Programms ermittelt die Sprachumgebung DTW_REXX, ob es OUT- oder INOUT-Funktionsparameter gibt. Trifft dies zu, ruft die Sprachumgebung den Wert, der dem Funktionsparameter entspricht, aus dem Variablenpool ab und aktualisiert den Funktionsparameterwert mit dem neuen Wert. Wenn Net.Data die Steuerung erhält, aktualisiert es alle OUT- oder INOUT-Parameter mit den neuen, aus der REXX-Sprachumgebung erhaltenen Werten. Beispiel:

```
%DEFINE a = "3"
%DEFINE b = "0"
%FUNCTION(DTW_REXX) double_func(IN inp1, OUT outp1){
    outp1 = 2*inp1
}%

%HTML(REPORT){
Value of b is $(b), @double_func(a, b) Value of b is $(b)
%}
```

Im obigen Beispiel übergibt der Aufruf *@double_func* zwei Parameter, *a* und *b*. Die REXX-Funktion *double_func* verdoppelt den ersten Parameter und speichert das Ergebnis im zweiten Parameter. Wenn Net.Data das Makro aufruft, hat *b* den Wert 6.

Sie können Net.Data-Tabellen an ein REXX-Programm übergeben. Ein REXX-Programm greift auf die Werte eines Net.Data-Makrotabellenparameters als REXX-Stammvariablen zu. Für ein REXX-Programm sind die Spaltenüberschriften und Feldwerte in Variablen enthalten, die mit dem Tabellennamen und der Spaltennummer angegeben werden. Zum Beispiel sind in der Tabelle *myTable* die Spaltenüberschriften *myTable_N.j* und die Feldwerte *myTable_N.i.j*, wobei *i* die Zeilennummer und *j* die Spaltennummer ist. Die Anzahl der Zeilen in der Tabelle ist *myTable_ROWS*, die Anzahl der Spalten *myTable_COLS*.

Beispiel für die REXX-Sprachumgebung

Das folgende Beispiel zeigt ein Makro, das eine REXX-Funktion aufruft, um eine Net.Data-Tabelle zu generieren, die zwei Spalten und drei Zeilen enthält. Nach dem Aufruf der REXX-Funktion wird eine integrierte Funktion, DTW_TB_TABLE(), aufgerufen, um eine HTML-Tabelle zu generieren, die an den Browser zurückgesendet wird.

```
%DEFINE myTable = %TABLE
%DEFINE DTW_DEFAULT_REPORT = "NO"

%FUNCTION(DTW_REXX) genTable(out out_table) {
  out_table_ROWS = 3
  out_table_COLS = 2

  /* Set Column Headings */
  do j=1 to out_table_COLS
    out_table_N.j = 'COL'j
  end

  /* Set the fields in the row */
  do i = 1 to out_table_ROWS
    do j = 1 to out_table_COLS
      out_table_V.i.j = '[' i j ']'
    end
  end
}

%HTML(REPORT){
  @genTable(myTable)
  @DTW_TB_TABLE(myTable)
}
```

Ergebnisse:

COL1	COL2
[1 1]	[1 2]
[2 1]	[2 2]
[3 1]	[3 2]

SQL-Sprachumgebung

Mit der SQL-Sprachumgebung können Sie SQL-Anweisungen ausführen, indem Sie die SQL-Anweisungen an ein Datenbankverwaltungssystem (DBMS) senden.

Stellen Sie sicher, daß Sie die in „Einrichten der Sprachumgebungen“ auf Seite 22 dokumentierten Konfigurationsschritte ausführen, bevor Sie die SQL-Sprachumgebung verwenden.

Ausführen von SQL-Anweisungen

Sie können eine beliebige SQL-Anweisung ausführen, die von dynamischem SQL unterstützt wird.

Definieren Sie zur Ausführung von SQL-Anweisungen eine Funktion, die die SQL-Sprachumgebung (DTW_SQL) verwendet und die SQL-Anweisungen im ausführbaren Abschnitt für die Sprachumgebung der Funktion enthält.

Beispiel: Eine SQL-Funktion, die eine SQL-Anweisung SELECT ausführt:

```
%function(DTW_SQL) getOrders() {  
    SELECT cust, custid, custorder FROM mylibrary.customers  
%}
```

COMMIT-Steuerung

Die SQL-Sprachumgebung wird standardmäßig unter COMMIT-Steuerung ausgeführt und befolgt alle Regeln zur Verwaltung der COMMIT-Steuerung.

- Aufzeichnen aller Dateien oder Tabellen, auf die über DTW_SQL zugegriffen wird, in einem Journal, es sei denn, die SQL-Anweisung lautet SELECT.
- Wahlfreie Änderung der COMMIT-Stufe, indem DTW_SQL_ISOLATION in der Net.Data-Initialisierungsdatei angegeben wird. Nähere Informationen zu den Isolationsstufen, die die SQL-Sprachumgebung unterstützt, finden Sie in „DTW_SQL_ISOLATION: DB2-Isolationsvariable“ auf Seite 12.

Weitere Informationen zur Transaktionsverwaltung finden Sie in „Verwalten von Transaktionen in einer Net.Data-Anwendung“ auf Seite 110.

OUT- und INOUT-Tabellen

Wenn Sie OUT- oder INOUT-Net.Data-Tabellen in der Funktionsdefinition angeben und die SQL-Anweisung Ergebnismengen zurückgibt, speichert die SQL-Sprachumgebung jede Ergebnismenge in den angegebenen Tabellen. Sie können die Tabelle dann später im Makro verwenden. Wird keine OUT-Tabelle angegeben, verwendet die SQL-Sprachumgebung eine Standardtabelle.

Verschachtelte SQL-Anweisungen

Sie können andere SQL-Funktionen von einem ROW-Block einer anderen SQL-Funktion aus aufrufen. Verwenden Sie eindeutige Net.Data-Tabellennamen in allen SQL-Funktionen. Ansonsten kann es zu unvorhersehbaren Ergebnissen kommen.

Beispiel: Aufrufen einer SQL-Funktion vom ROW-Block einer anderen SQL-Funktion aus

```
%define mytable1 = %TABLE  
%define mytable2 = %TABLE  
  
%FUNCTION(DTW_SQL) sql2 (IN p1, OUT t2) {  
    select * from NETDATA.STAFFINF where projno='$(p1)'  
%REPORT {  
    %ROW { $(N1) is $(V1) %}  
%}  
%}  
  
%FUNCTION(DTW_SQL) sql1 (OUT t1) {  
    select * from NETDATA.STAFFINF  
%REPORT {  
    %ROW { @sql2(V1, mytable2) %}  
%}  
%}  
  
%HTML(netcall1) { @sql1(mytable1) %}
```

Unterstützte Sprachumgebungsvariablen

Die SQL-Sprachumgebung unterstützt Variablen zur Unterstützung von DB2. Zum Beispiel gibt die Variable DATABASE die Datenquelle an, zu der die SQL-Sprachumgebung die Verbindung herstellt, wenn eine SQL-Anweisung ausgeführt wird. In der folgenden Liste wird angegeben, welche Variablen für die SQL-Sprachumgebung unterstützt werden. Im Handbuch *Net.Data Reference* finden Sie eine Beschreibung, eine Syntaxdarstellung und Beispiele für diese Variablen.

- DATABASE
- DB_CASE
- DTW_EDIT_CODES
- DTW_PAD_PGM_PARMS
- DTW_SET_TOTAL_ROWS
- LOGIN
- NULL_RPT_FIELD
- PASSWORD
- SHOWSQL
- SQL_STATE
- TRANSACTION_SCOPE

Unterstützte Datentypen

Die SQL-Sprachumgebung unterstützt die in Tabelle 5 aufgelisteten Datentypen.

Tabelle 5. Datentypen

BLOB(1)	DOUBLE	SMALLINT
CHAR	DOUBLEPRECISION	TIME
CLOB(1)	FLOAT	TIMESTAMP
DATE	GRAPHIC	VARCHAR
DBCLOB(1)	INTEGER	VARGRAPHIC
DECIMAL	REAL	

(1) Diese Datentypen können nicht als Parameter an einen gespeicherten Prozeduraufruf übergeben werden. Informationen dazu, welche Datentypen für gespeicherte Prozeduren unterstützt werden, finden Sie in „Syntax für gespeicherte Prozeduren“ auf Seite 112.

In „Überlegungen zu Datentypen“ auf Seite 105 finden Sie besondere Überlegungen für die Datentypen LOB und DATALINK.

Einschränkungen der SQL-Sprachumgebung

Berücksichtigen Sie die folgenden Einschränkungen, wenn Sie Ihre Umgebung planen:

- Verwenden Sie die SQL-Sprachumgebung nicht, wenn eine oder mehrere der folgenden Bedingungen vorliegen:
 - Es wird eine benutzerdefinierte Sprachumgebung erstellt, die die Klassenbibliothek für Datenbankzugriff oder die SQL Call Level Interface (CLI) verwendet, und auf die benutzerdefinierte Sprachumgebung wird in einem Makro verwiesen.
 - Eine Anwendung, die die SQL-CLI verwendet, wird im gleichen Prozeß wie Net.Data ausgeführt.
- SQL-Anweisungen im Inline-Anweisungsblock können bis zu 32 KB groß sein.
- Sie können bis zu 50 Verbindungen zu lokalen oder fernen Datenbanken verwenden. Beachten Sie bei der Verwendung von Mehrfachverbindungen die folgenden Einschränkungen:
 - Net.Data erlaubt keine gleichzeitig bestehenden Verbindungen zu der gleichen fernen Datenbank.
 - Sie können die Anmelde-IDs nicht ändern, nachdem Sie auf eine ferne Datenbank zugegriffen haben, wenn TRANSACTION_SCOPE=MULTIPLE (Standardeinstellung) definiert ist. Siehe „Verwalten von Transaktionen in einer Net.Data-Anwendung“ auf Seite 110.

Weitere Informationen zu diesen Einschränkungen finden Sie in „Verwalten von Mehrfachdatenbankverbindungen“ auf Seite 111.

Überlegungen zu Datentypen

Die folgenden von der SQL-Sprachumgebung unterstützten Datentypen müssen gesondert betrachtet werden.

- „Verwenden großer Objekte“
- „Codieren von DataLink-URL-Adressen in Ergebnismengen“ auf Seite 109

Verwenden großer Objekte

Sie können Dateien großer Objekte (LOBs) in DB2-Datenbanken speichern und mit der SQL-Sprachumgebung für Ihre Web-Anwendungen auf sie zugreifen.

Die SQL-Sprachumgebung speichert große Objekte weder in Net.Data-Variablen zur Tabellenverarbeitung (wie V1 oder V2) noch in Net.Data-Tabellenfeldern, wenn eine SQL-Abfrage LOBs in einer Ergebnismenge zurückgibt. Anstelle dessen wird das große Objekt nach seiner Feststellung durch Net.Data in einer von Net.Data erstellten Datei gespeichert. Diese Datei befindet sich in einem durch die Pfadkonfigurationsvariable HTML_PATH angegebenen Verzeichnis. Die Werte der Net.Data-Tabellenfelder und -Variablen zur Tabellenverarbeitung werden auf den Pfad zu dieser Datei gesetzt. Der Zugriff auf die Datei ist auf die Benutzer-ID beschränkt, unter der Net.Data ausgeführt wird.

Der Dateiname, unter dem das große Objekt gespeichert wird, wird dynamisch erstellt und weist folgendes Format auf:

name[*.extension*]

Dabei gilt folgendes:

name Dies ist eine eindeutige Zeichenfolge, die das große Objekt angibt.

extension Dies ist eine Zeichenfolge, die den Objekttyp angibt. Bei CLOBs und DBCLOBs ist die Erweiterung 'txt'. Bei BLOBs versucht die SQL-Sprachumgebung, die Erweiterung zu ermitteln, indem sie in den ersten Byte der LOB-Datei nach einer Kennung sucht, die angibt, was das große Objekt darstellt. Die SQL-Sprachumgebung erkennt die folgenden Datentypen (in den runden Klammern steht die verwendete Erweiterung):

- Bitmap-Abbild (.bmp)
- Graphical Image Format (.gif)
- JPEG-Abbilddateien (.jpg)
- Tagged Image File Format (.tif)
- PostScript (.ps)
- MIDI-Audiodatei (.mid)
- AIFF-Audiodatei (.aif)
- AVI-Datei (.avi)
- Basisaudiodateien (.au)
- RA-Dateien (.ra)
- WAV-Dateien (.wav)
- Portable Document Format (.pdf)
- MIDI-Sequenzdatei (.rmi)

Wird der Objekttyp des binären großen Objekts (BLOB) nicht erkannt, wird dem Dateinamen keine Erweiterung hinzugefügt.

Wenn in der Makrodatei auf das große Objekt verwiesen wird, gibt die SQL-Sprachumgebung den Dateinamen zurück und stellt ihm die Zeichenfolge /tmplobs/ vor. Dabei wird folgende Syntax verwendet:

/tmplobs/*name*.*[extension]*

Hinweis zur Planung: Jede Abfrage, die LOBs zurückgibt, erstellt im durch die Pfadkonfigurationsvariable HTML_PATH angegebenen Verzeichnis Dateien. Bei der Verwendung von LOBs ist die jeweilige Systemausstattung zu berücksichtigen, da diese Objekte die Systemressourcen schnell aufbrauchen können. Erwägen Sie das Erstellen eines Stapelverarbeitungsprogramms, das das Verzeichnis periodisch aufräumt. Es wird empfohlen, DataLinks zu verwenden. Dadurch braucht die SQL-Sprachumgebung Dateien nicht mehr in Verzeichnissen zu speichern, was zu einer Leistungsoptimierung und einer wesentlichen Reduzierung der belegten Systemressourcen führt.

Beispiel: Der Anwendungsbenutzer muß den Dateinamen anklicken, um die Anzeigefunktion aufzurufen, weil die Anwendung eine MPEG-Audiodatei (.MPA) verwendet. Die SQL-Sprachumgebung erkennt diesen Dateityp nicht. Daher wird die Erweiterung mit einer EXEC-Variablen an die Datei angehängt.

```
%DEFINE{
lobpath = "@DTW_RGETINIDATA("HTML_PATH")"
filename = "@DTW_RREPLACE($(V3), "/tmplobs/", "", "1", "F")"
myFile=%EXEC "REN '$(lobpath)/$(filename)' '$(filename).mpa'"
%}

%{ where rename is the rename command on your operating system %}
%FUNCTION(DTW_SQL) queryData() {
SELECT Name, IDPhoto, Voice FROM RepProfile
  %REPORT{
    <P>Here is the information you selected:<P>
    %ROW{
      $(myFile)
      $(V1) Voice sample <IMG SRC="$(V2)">
      <A HREF="$(V3).mpa">Voice sample</A><P>
    %}
    %}
    %}

%HTML(REPORT){
@queryData()
%}
```

Die Funktion queryData gibt folgende HTML-Ausgabe zurück:

```
<P>Here are the images you selected:<P>
Kinson Yamamoto
<IMG SRC="/tmplobs/p2345n1.gif">
<A HREF="/tmplobs/p2345n2.mpa">Voice sample</A><P>
Merilee Lau
<IMG SRC="/tmplobs/p2345n3.gif">
<A HREF="/tmplobs/p2345n4.mpa">Voice sample</A><P>
```

Der REPORT-Block im vorangegangenen Beispiel verwendet die impliziten Tabellenvariablen V1, V2 und V3.

- V1 ist der Name einer Person. Dabei handelt es sich um unverschlüsselten Text.
- V2 ist das Foto der Person in einer GIF-Datei. Das Bild wird inline angezeigt. Die SQL-Sprachumgebung enthält automatisch das Präfix /tmplobs/ und die Erweiterung GIF.
- V3 ist ein Beispiel für ein Stimmuster der Person in einer MPA-Datei. Wenn die SQL-Sprachumgebung ein unerkanntes Format wie zum Beispiel eine MPA-Datei vorfindet, schreibt sie die Datei ohne Dateierweiterung in das durch die Konfigurationsvariable HTML_PATH angegebene Verzeichnis. Dieses Beispiel zeigt, wie ein solcher Dateityp durch Hinzufügen der Erweiterung mit einer EXEC-Variablen verwendet werden kann. Wenn die Variable \$(V3) aufgelöst wird, wird vor dem Dateinamen der Pfad /tmplobs/ eingefügt, z. B. /tmplobs/sound2a. Im obigen Beispiel benennt die Variable EXEC die Datei mit dem Befehl REN um und fügt der Datei die Erweiterung .mpa hinzu. Bevor der Dateiname umbenannt werden kann, muß /tmplobs/ aus dem Dateinamen entfernt werden und muß der vollständige Pfad zu der umzubenennenden Datei abgerufen werden. Hierzu ruft die Funktion DTW_RGETINIDATA den in HTML_PATH angegebenen Pfad ab. Das Stimmuster wird wiedergegeben, wenn der Anwendungsbenutzer Voice sample anklickt.

Zugriffsrechte für LOBs: Stellen Sie sicher, daß die Benutzer-ID, unter der der Web-Server ausgeführt wird, Schreibzugriff auf das durch HTML_PATH angegebene Verzeichnis hat.

Codieren von DataLink-URL-Adressen in Ergebnismengen

Der Datentyp DATALINK ist einer der Grundbausteine für die Erweiterung der Datentypen, die in Datenbankdateien gespeichert werden können. Bei DATALINK sind die in der Spalte gespeicherten Daten lediglich ein Zeiger zur Datei. Diese Datei kann in einem beliebigen Dateityp vorliegen, z. B. eine Abbilddatei, eine Stimmzeichnung oder eine Textdatei. DataLink-Datentypen speichern eine URL-Adresse, um die Speicherposition der Datei aufzulösen.

Für den Datentyp DATALINK ist die Verwendung von DataLink File Manager erforderlich. Weitere Informationen zu DataLink File Manager finden Sie in der DataLinks-Dokumentation für Ihr Betriebssystem. Vor der Verwendung des Datentyps DATALINK müssen Sie sicherstellen, daß der Web-Server Zugriff auf das durch den Server mit DB2 File Manager verwaltete Dateisystem hat.

Wenn eine SQL-Abfrage eine Ergebnismenge mit DataLinks zurückgibt und die DataLink-Spalte mit den DataLink-Optionen FILE LINK CONTROL und READ PERMISSION DB erstellt wird, enthalten die Dateipfade in der DataLink-Spalte ein Zugriffs-Token. DB2 überprüft mit dem Zugriffs-Token den Zugriff auf die Datei. Ohne dieses Zugriffs-Token schlagen alle Zugriffsversuche auf die Datei mit einer Berechtigungsverletzung fehl. Das Zugriffs-Token enthält jedoch unter Umständen Zeichen, die in einer URL-Adresse (die an einen Browser zurückgegeben werden soll) nicht verwendet werden können, z. B. das Semikolon (;). Beispiel:

```
/datalink/pics/UN1B;0YPVKG346KEBE;baibien.jpg
```

Die URL-Adresse ist ungültig, weil sie Semikolons (;) enthält. Die Semikolons müssen mit der integrierten Net.Data-Funktion DTW_URLESCSEQ codiert werden, um die Ungültigkeit der URL-Adresse aufzuheben. Vor der Anwendung dieser Funktion muß die Zeichenfolge allerdings noch bearbeitet werden, weil diese Funktion auch Schrägstriche (/) codiert.

Sie können eine Net.Data-Makrofunktion (MACRO_FUNCTION) schreiben, um die Bearbeitung der Zeichenfolge zu automatisieren und die Funktion DTW_URLESCSEQ zu verwenden. Verwenden Sie dieses Verfahren in jedem Makro, das Daten aus einer Spalte mit dem Datentyp DATALINK abruft.

Beispiel 1: MACRO_FUNCTION zur Automatisierung der Codierung von URL-Adressen, die von DB2 UDB zurückgegeben werden

```
%{ TO DO: Apply DTW_URLESCSEQ to a DATALINK URL to make it a valid URL.
   IN: DATALINK URL from DB2 File Manager column.
   RETURN: The URL with token portion is URL encoded
%}
%MACRO_FUNCTION encodeDataLink(in DLURL) {
  @DTW_rCONCAT( @DTW_rDELSTR( DLURL,
    @DTW_rADD(@DTW_rLASTPOS("/", DLURL), "1" ) ),
    @DTW_rURLESCSEQ( @DTW_rSUBSTR(DLURL,
      @DTW_rADD( @DTW_rLASTPOS("/", DLURL), "1" ) ) ) )
%}
```

Nach der Verwendung dieser Makrofunktion ist die URL-Adresse ordnungsgemäß codiert, und auf die in der Spalte DATALINK angegebene Datei kann in einem beliebigen Web-Browser verwiesen werden.

Beispiel 2: Ein Net.Data-Makro zur Angabe der SQL-Abfrage, die die DATALINK-URL-Adresse zurückgibt

```
%FUNCTION(DTW_SQL) myQuery(){
  select name, DLURLCOMPLETE(picture) from myTable where name like '%river%'
  %REPORT{
    %ROW{
      <p> $(V1) <br>
      Before Encoding: $(V2) <br>
      After Encoding: @encodeDataLink$(V2)) <br>
      Make HREF: <a href="@encodeDataLink$(V2))"> click here </a> <br> <p>
    %}
  %}
}
```

Beachten Sie, daß eine Funktion von DataLink File Manager verwendet wird. Die Funktion `dlurlcomplete` gibt eine vollständige URL-Adresse zurück.

Verwalten von Transaktionen in einer Net.Data-Anwendung

Wenn Sie den Inhalt einer Datenbank mit INSERT-, DELETE- oder UPDATE-Anweisungen ändern, werden diese Änderungen erst festgeschrieben, nachdem die Datenbank eine COMMIT-Anweisung von Net.Data empfangen hat. Wenn ein Fehler auftritt, sendet Net.Data eine ROLLBACK-Anweisung an die Datenbank, die alle Änderungen seit der letzten COMMIT-Operation zurücknimmt.

Wie Net.Data die COMMIT-Anweisung und die etwaige ROLLBACK-Anweisung sendet, hängt davon ab, wie Sie TRANSACTION_SCOPE festlegen und ob Sie die COMMIT-Anweisung explizit im Makro angeben. Zulässige Werte für TRANSACTION_SCOPE sind MULTIPLE und SINGLE.

MULTIPLE

Gibt an, daß Net.Data alle SQL-Anweisungen vor dem Absetzen einer COMMIT- und etwaigen ROLLBACK-Anweisung ausführt. Net.Data sendet die COMMIT-Anweisung am Ende der Anforderung, und wenn jede SQL-Anweisung erfolgreich abgesetzt wird, schreibt die COMMIT-Anweisung alle Änderungen in der Datenbank fest. Wenn durch eine der Anweisungen ein Fehler zurückgegeben wird, setzt Net.Data eine ROLLBACK-Anweisung ab, die die Datenbank auf ihren ursprünglichen Status zurücksetzt. MULTIPLE ist der Standardwert, wenn TRANSACTION_SCOPE nicht festgelegt wird.

Sie aktivieren diese COMMIT-Methode, indem Sie TRANSACTION_SCOPE auf MULTIPLE setzen.

Beispiel:

```
@DTW_ASSIGN(TRANSACTION_SCOPE,"MULTIPLE")
```

SINGLE

Gibt an, daß Net.Data nach jeder erfolgreichen SQL-Anweisung eine COMMIT-Anweisung absetzt. Wenn die SQL-Anweisung einen Fehler zurückgibt, wird eine ROLLBACK-Anweisung abgesetzt.

Die Angabe SINGLE für TRANSACTION_SCOPE stellt eine sofortige Datenbankänderung sicher. Allerdings kann eine Änderung später mit Hilfe einer ROLLBACK-Anweisung nicht widerrufen werden.

Sie aktivieren diese COMMIT-Methode, indem Sie TRANSACTION_SCOPE auf SINGLE setzen. Beispiel:

```
@DTW_ASSIGN(TRANSACTION_SCOPE,"SINGLE")
```

Sie können mit der SQL-Anweisung COMMIT am Ende jeder beliebigen SQL-Anweisung in Ihrem Makro eine COMMIT-Anweisung absetzen.

Wenn Sie als Anwendungsentwickler die Standardeinstellung MULTIPLE für TRANSACTION_SCOPE beibehalten und wenn Sie COMMIT-Anweisungen am Ende der Anweisungsgruppen absetzen, die als Transaktion in Frage kommen, haben Sie volle Kontrolle über das COMMIT- und ROLLBACK-Verhalten in Ihrer Anwendung.

Zum Absetzen einer SQL-Anweisung COMMIT können Sie eine Funktion definieren, die Sie an einem beliebigen Punkt in Ihrem HTML-Block aufrufen können:

```
%FUNCTION(DTW_SQL) user_commit() {  
    commit  
%}
```

```
...
```

```
%HTML {  
    ...  
    @user_commit()  
    ...  
%}
```

Verwalten von Mehrfachdatenbankverbindungen

Sie können Verbindungen zu bis zu 50 lokalen oder fernen Datenbanken gleichzeitig herstellen. Die SQL-Sprachumgebung erhält die Aktivierung der Verbindungen für die Dauer des Web-Server-Prozeßjobs, unter dem Net.Data ausgeführt wird, aufrecht. Hierdurch wird der Datenbankzugriff beschleunigt, nachdem die einleitende Verbindung zur Datenbank hergestellt wurde. Sie können Fehler vermeiden, indem Sie die folgenden Punkte berücksichtigen:

- Net.Data erlaubt keine gleichzeitig bestehenden Verbindungen zu der gleichen fernen Datenbank. Wenn eine Verbindung zu einer fernen Datenbank unter Verwendung einer Benutzer-ID (dem SQL-Sprachumgebungsparameter LOGIN) besteht und eine weitere Anforderung abgesetzt wird, um eine Verbindung zu der gleichen fernen Datenbank unter Verwendung einer zweiten Benutzer-ID herzustellen, muß die SQL-Sprachumgebung zuerst die bestehende Verbindung trennen, eine COMMIT-Operation ausführen (wenn die COMMIT-Steuerung verwendet wird) und dann die Verbindung unter Verwendung der neuen Benutzer-ID und des neuen Kennworts erneut herstellen. Die COMMIT-Operation ist erforderlich, da es nach Trennen der Verbindung nicht möglich ist, eine ROLLBACK-Operation auszuführen, wenn später im Makro ein Fehler auftritt.

- Sie können die Anmelde-ID wechseln, nachdem Sie auf eine ferne Datenbank zugegriffen haben, wenn TRANSACTION_SCOPE=SINGLE definiert ist. Die SQL-Sprachumgebung trennt die bestehende Verbindung, führt eine COMMIT-Operation aus und stellt die Verbindung mit der neuen Benutzer-ID und dem neuen Kennwort erneut her.
- Wechseln Sie die Anmelde-ID nicht, nachdem Sie auf eine ferne Datenbank zugegriffen haben, wenn TRANSACTION_SCOPE=MULTIPLE (Standardeinstellung) definiert ist. Die SQL-Sprachumgebung führt automatisch eine ROLLBACK-Operation aus, und der SQL_CODE -752 wird zurückgegeben, der darauf hinweist, daß die Verbindung nicht geändert werden konnte.

Gespeicherte Prozeduren

Eine gespeicherte Prozedur ist ein kompiliertes Programm, das in DB2 gespeichert ist und SQL-Anweisungen ausführen kann. In Net.Data werden gespeicherte Prozeduren von Net.Data-Funktionen mit Hilfe der Anweisung CALL aufgerufen. Parameter für gespeicherte Prozeduren werden aus der Parameterliste der Net.Data-Funktion übergeben.

Sie können gespeicherte Prozeduren einsetzen, um die Leistung und die Integrität zu verbessern, indem Sie kompilierte SQL-Anweisungen auf dem Datenbank-Server speichern. Net.Data unterstützt die Verwendung gespeicherter Prozeduren unter DB2 über die SQL- und ODBC-Sprachumgebungen.

In diesem Abschnitt werden folgende Themen behandelt:

- „Syntax für gespeicherte Prozeduren“
- „Aufrufen einer gespeicherten Prozedur“ auf Seite 113
- „Übergeben von Parametern“ auf Seite 115
- „Verarbeiten von Ergebnismengen“ auf Seite 115

Syntax für gespeicherte Prozeduren

Die Syntax für die gespeicherte Prozedur verwendet die Anweisung FUNCTION, die Anweisung CALL und wahlfrei einen REPORT-Block.

```
%FUNCTION function_name ([IN datatype arg1, INOUT datatype arg2,
    OUT tablename, ...]) {
    CALL stored_procedure
[%REPORT [(resultsetname)] { %}]
...
[%REPORT [(resultsetname)] { %}]
[%MESSAGE %]}

%}
```

Dabei gilt folgendes:

function_name

Ist der Name der Net.Data-Funktion, die den Aufruf der gespeicherten Prozedur initialisiert.

stored_procedure

Ist der Name der gespeicherten Prozedur.

datatype

Ist einer der von Net.Data unterstützten Datentypen der Datenbank wie in Tabelle 6 gezeigt. Die in der Parameterliste angegebenen Datentypen müssen mit den Datentypen in der gespeicherten Prozedur übereinstimmen. Weitere Informationen zu diesen Datentypen finden Sie in Ihrer Datenbankdokumentation.

tablename

Ist der Name einer Net.Data-Tabelle, in der die Ergebnismenge gespeichert werden soll (wird nur verwendet, wenn die Ergebnismenge in einer Net.Data-Tabelle gespeichert werden soll). Wenn dieser Parametername angegeben ist, muß er mit dem zugehörigen Parameternamen für *resultsetname* übereinstimmen.

resultsetname

Ist der Name, der ein von einer gespeicherten Prozedur zurückgegebenes Ergebnis einem REPORT-Block und/oder einem Tabellennamen in der Funktionsparameterliste zuordnet. Die Angabe *resultsetname* in einem REPORT-Block muß mit einer Angabe für *tablename* in der Funktionsparameterliste übereinstimmen.

Tabelle 6. Datentypen für gespeicherte Prozeduren

CHAR	FLOAT	SMALLINT
DATE	GRAPHIC	TIME
DECIMAL	INTEGER	TIMESTAMP
DOUBLE	REAL	VARCHAR
DOUBLEPRECISION		VARGRAPHIC

Aufrufen einer gespeicherten Prozedur

1. Definieren Sie eine Funktion, die einen Aufruf an die gespeicherte Prozedur initialisiert.

```
%FUNCTION (DTW_SQL) function_name()
```

2. Geben Sie wahlfrei IN-, INOUT- oder OUT-Parameter für die gespeicherte Prozedur an, einschließlich des Namens von Ergebnismengen, die von der gespeicherten Prozedur zurückgegeben werden. Sie können für die Tabellennamen bzw. Ergebnismengen auch IN- oder INOUT-Parameter aus einer anderen gespeicherten Prozedur angeben.

```
%FUNCTION (DTW_SQL) function_name (IN datatype  
arg1, INOUT datatype arg2,  
OUT tablename...)
```

3. Geben Sie mit Hilfe der Anweisung CALL den Namen der gespeicherten Prozedur an.

```
CALL stored_procedure
```

4. Wenn die gespeicherte Prozedur nur eine Ergebnismenge generiert, können Sie wahlfrei einen REPORT-Block angeben, um zu definieren, wie Net.Data die Ergebnismenge anzeigen soll.

```
%REPORT {  
...  
%}
```

Beispiel:

```
%FUNCTION (DTW_SQL) mystoredproc (IN CHAR(30) arg1 OUT mytable) {  
    CALL myproc  
    %REPORT {  
        ...  
        %ROW { ... %}  
        ...  
    %}  
%}
```

5. Wenn die gespeicherte Prozedur mehrere Ergebnismengen generiert, haben Sie folgende Möglichkeiten:

- Geben Sie die Ergebnismengen als OUT-Parameter in der FUNCTION-Anweisung an. Die Ergebnismengen werden als lokale Tabellen gespeichert.

```
%FUNCTION (DTW_SQL) function_name (OUT tablename, ...)
```

- Geben Sie wahlfrei einen oder mehrere REPORT-Blöcke an, um zu definieren, wie Net.Data die Ergebnismengen anzeigen soll.

```
%REPORT(resultsetname1) {  
...  
%}
```

Beispiel:

```
%FUNCTION (DTW_SQL) mystoredproc (IN CHAR(30) arg1, OUT table1,  
    table2) { CALL myproc  
    %REPORT(table1) {  
        ...  
        %ROW { ... %}  
        ...  
    %}  
    %REPORT(table1) {  
        ...  
        %ROW { ... %}  
        ...  
    %}  
%}
```

Übergeben von Parametern

Sie können Parameter an eine gespeicherte Prozedur übergeben und die gespeicherte Prozedur die Parameterwerte aktualisieren lassen, so daß die neuen Werte an das Net.Data-Makro zurückgegeben werden. Die Anzahl und der Typ der Parameter in der Funktionsparameterliste müssen mit der Anzahl und dem Typ übereinstimmen, die für die gespeicherte Prozedur definiert sind. Wenn z. B. ein Parameter in der für die gespeicherte Prozedur definierten Parameterliste INOUT ist, dann muß der entsprechende Parameter in der Funktionsparameterliste auch INOUT sein. Wenn ein Parameter in der für die gespeicherte Prozedur definierten Parameterliste vom Typ CHAR(30) ist, dann muß der entsprechende Parameter in der Funktionsparameterliste auch CHAR(30) sein.

Beispiel 1: Übergeben eines Parameterwerts an die gespeicherte Prozedur

```
%FUNCTION (DTW_SQL) mystoredproc (IN CHAR(30) valuein) {  
    CALL myproc  
    ...  
}
```

Beispiel 2: Zurückgeben eines Werts aus einer gespeicherten Prozedur

```
%FUNCTION (DTW_SQL) mystoredproc (OUT VARCHAR(9) retvalue) {  
    CALL myproc  
    ...  
}
```

Verarbeiten von Ergebnismengen

Sie können eine oder mehrere Ergebnismengen aus einer gespeicherten Prozedur zurückgeben. Die Ergebnismengen können in Net.Data-Tabellen zur weiteren Verarbeitung innerhalb Ihres Makros gespeichert oder mit Hilfe eines REPORT-Blocks verarbeitet werden. Wenn eine gespeicherte Prozedur mehrere Ergebnismengen generiert, müssen Sie jeder durch die gespeicherte Prozedur generierten Ergebnismenge einen Namen zuordnen. Dies geschieht durch die Angabe von Parametern in der Anweisung FUNCTION. Der Name, den Sie für eine Ergebnismenge angeben, kann anschließend einem REPORT-Block oder einer Net.Data-Tabelle zugeordnet werden, so daß Sie festlegen können, wie die einzelnen Ergebnismengen von Net.Data verarbeitet werden. Sie haben folgende Möglichkeiten:

- Sie können das Ergebnis in der Standarddarstellung für Net.Data-Berichte verarbeiten, indem Sie keinen REPORT-Block für die Ergebnismenge definieren.
- Sie können eine Ergebnismenge einem REPORT-Block zuordnen, um Ihre eigene Berichtsdarstellung anzuwenden. Sie können im REPORT-Block mit Net.Data-Variablen, -Textverarbeitungsanweisungen wie HTML oder JavaScript oder anderen Funktionen angeben, wie die Berichtsdaten vom Browser angezeigt werden sollen.

Ergebnismengen werden immer in lokalen Tabellen gespeichert, so daß eine andere Funktion im Makro auch auf die Daten zugreifen kann. Zum Beispiel können Sie die Net.Data-Tabelle an eine andere Funktion übergeben, die die Daten zu Berechnungen und zum Anzeigen der berechneten Ergebnisse verwenden kann.

Richtlinien und Einschränkungen zur Verwendung mehrerer REPORT-Blöcke finden Sie in „Richtlinien und Einschränkungen für mehrere REPORT-Blöcke“ auf Seite 85.

Gehen Sie wie folgt vor, wenn nur eine Ergebnismenge zurückgegeben und diese in einem Standardbericht angezeigt werden soll:

Verwenden Sie die folgende Syntax:

```
%FUNCTION (DTW_SQL) function_name (OUT tablename) {  
    CALL stored_procedure  
%}
```

Beispiel:

```
%FUNCTION (DTW_SQL) mystoredproc(OUT mytable1) {  
    CALL myproc  
%}
```

Gehen Sie wie folgt vor, wenn nur eine Ergebnismenge zurückgegeben und ein REPORT-Block angegeben werden soll:

Verwenden Sie die folgende Syntax:

```
%FUNCTION (DTW_SQL) function_name (OUT tablename) {  
    CALL stored_procedure [(resultsetname)]  
    %REPORT [(resultsetname)] {  
        ...  
    %}  
%}
```

Beispiel:

```
%FUNCTION (DTW_SQL) mystoredproc (OUT mytable1) {  
    CALL myproc  
    %REPORT {  
        ...  
        %ROW { ... %}  
        ...  
    %}  
%}
```

Alternativ kann die folgende Syntax verwendet werden:

```
%FUNCTION (DTW_SQL) function_name () {  
    CALL stored_procedure  
  
    %REPORT () {  
        ...  
    %}  
%}
```

Beispiel:

```
%FUNCTION (DTW_SQL) mystoredproc () {  
    CALL myproc  
    %REPORT {  
        ...  
        %ROW { ... %}  
        ...  
    %}  
%}
```


Gehen Sie wie folgt vor, wenn mehrere Ergebnismengen zurückgegeben und diese im Standardberichtsformat angezeigt werden sollen:

Verwenden Sie die folgende Syntax:

```
%FUNCTION (DTW_SQL) function_name (OUT tablename1, tablename2) {  
    CALL stored_procedure  
%}
```

Dabei wird kein REPORT-Block angegeben.

Beispiel:

```
%DEFINE DTW_DEFAULT_REPORT = "YES"  
%FUNCTION (DTW_SQL) mystoredproc (OUT mytable1, mytable2) {  
    CALL myproc  
%}
```

Gehen Sie wie folgt vor, wenn mehrere Ergebnismengen zurückgegeben und REPORT-Blöcke für die Verarbeitung der Anzeige angegeben werden sollen:

Jede Ergebnismenge ist einem REPORT-Block oder mehreren REPORT-Blöcken zugeordnet. Verwenden Sie die folgende Syntax:

```
%FUNCTION (DTW_SQL) function_name (OUT tablename1, tablename2, ...) {  
    CALL stored_procedure  
    %REPORT (tablename1)  
        ...  
    %ROW { ... %}  
        ...  
    %}  
    %REPORT (tablename2)  
        ...  
    %ROW { ... %}  
        ...  
    %}  
    ...  
%}
```

Beispiel:

```
%FUNCTION (DTW_SQL) mystoredproc (OUT mytable1, mytable2) {  
    CALL myproc  
  
    %REPORT (mytable1) {  
        ...  
    %ROW { ... %}  
        ...  
    %}  
  
    %REPORT(mytable2) {  
        ...  
    %ROW { ... %}  
        ...  
    %}  
%}
```

Beispiel für die SQL-Sprachumgebung

Das folgende Beispiel zeigt ein Makro mit einer DTW_SQL-Funktionsdefinition, die eine gespeicherte SQL-Prozedur aufruft. Es enthält drei Parameter mit unterschiedlichen Datentypen. Die Sprachumgebung DTW_SQL wandelt die Zeichenfolgewerte in jedem Parameter in das richtige interne Format um und übergibt jeden Parameter nach Verweis an die gespeicherte SQL-Prozedur. Wenn die Verarbeitung der gespeicherten SQL-Prozedur beendet ist, wird die aktualisierte interne Darstellung in eine Zeichenfolge umgewandelt und in den entsprechenden Parameter gestellt.

```
%{*****
*****DEFINE BLOCK*****
*****%}
DEFINE {
MACRO_NAME      = "TEST ALL TYPES"
DTW_HTML_TABLE  = "YES"
Procedure       = "NDLIB.TESTTYPE"
parm1           = "1"                %{SMALLINT           %}
parm2           = "11"               %{INT                 %}
parm3           = "1.1"              %{DECIMAL (2,1)        %}
%}

%FUNCTION(DTW_SQL) CRTPROC(){
CREATE PROCEDURE $(Procedure)
( INOUT SMALLINT,
  INOUT INT,
  INOUT DECIMAL(2,1))
EXTERNAL NAME $(Procedure) LANGUAGE C SIMPLE CALL
%MESSAGE{
  default : "$(DTW_DEFAULT_MESSAGE) : continuing.<br>": continue
%}
%}

%FUNCTION(DTW_SQL)    myProc
  (INOUT SMALLINT      parm1,
   INOUT INT           parm2,
   INOUT DECIMAL(2,1)  parm3){
CALL $(Procedure)
%}
```

```

%HTML(REPORT){
<HEAD>
<TITLE>Net.Data : SQL Stored Procedure: Example '$(MACRO_NAME)'. <?TITLE>
</HEAD>
<BODY BGCOLOR="#BBFFFF" TEXT="#000000" LINK="#000000">
<p><p>
Calling the function to create the stored procedure.
<p><p>
    @CRTPROC()
<hr>
<h2>
Values of the INOUT parameters
prior to calling the stored procedure:<p>
</h2>
<b>parm1 (SMALLINT)</b><br>
$(parm1)<p>
<b>parm2 (INT)</b><br>
$(parm2)<p>
<b>parm3 (DECIMAL)</b><br>
$(parm3)<p>
<p>
<hr>
<h2>
Calling the function that executes the stored procedure.
</h2>
<p><p>
    @myProc(parm1,parm2,parm3)
<hr>
<h2>
Values of the INOUT parameters after
calling the stored procedure:<p>
</h2>
<b>parm1 (SMALLINT)</b><br>
$(parm1)<p>
<b>parm2 (INT)</b><br>
$(parm2)<p>
<b>parm3 (DECIMAL)</b><br>
$(parm3)<p>
</body>
%}

```

SYSTEM-Sprachumgebung

Die SYSTEM-Sprachumgebung unterstützt das Ausführen von Befehlen und das Aufrufen externer Programme.

Absetzen von Befehlen und Aufrufen von Programmen

Definieren Sie zum Absetzen eines Befehls eine Funktion, die die SYSTEM-Sprachumgebung (DTW_SYSTEM) verwendet und einen Pfad zu dem Befehl enthält, der in einer EXEC-Anweisung abgesetzt werden soll. Beispiel:

```
%FUNCTION(DTW_SYSTEM) sys1() {  
    %EXEC { /QSYS.LIB/ADDLIBLE.CMD LIB(MYLIBRARY) %}  
    %}
```

Sie können den Pfad zu ausführbaren Objekten kürzen, wenn Sie die Konfigurationsvariable EXEC_PATH zum Definieren von Pfaden zu Verzeichnissen mit den Objekten (wie Befehle und Programme) verwenden. Informationen zum Definieren der Konfigurationsvariablen EXEC_PATH finden Sie in „EXEC_PATH“ auf Seite 17.

Beispiel 1: Absetzen eines Befehls

```
%FUNCTION(DTW_SYSTEM) sys2() {  
    %EXEC { /QSYS.LIB/CALL.CMD MYLIB/MYPGM %}  
    %}
```

Beispiel 2: Aufrufen eines Programms

```
%FUNCTION(DTW_SYSTEM) sys3() {  
    %EXEC { /QSYS.LIB/MYLIB.LIB/MYPGM.PGM %}  
    %}
```

Tip: Verwenden Sie beim Aufrufen von Programmen die Direktaufsprachumgebung, weil sie effektiver und einfacher zu verwenden ist.

Übergeben von Parametern an Programme

Es gibt zwei Möglichkeiten, Informationen an ein Programm zu übergeben, das durch die SYSTEM-Sprachumgebung (DTW_SYSTEM) aufgerufen wird: direkt und indirekt.

Direkt Sie übergeben Parameter direkt beim Aufruf des Programms. Beispiel:

```
%DEFINE INPARAM1 = "SWITCH1"  
  
%FUNCTION(DTW_SYSTEM) sys1() {  
    %EXEC {  
        /QSYS.LIB/NETDATA.LIB/RPGCALL1.PGM ('${INPARAM1}' 'literalstring')  
    %}  
    %}
```

Die Net.Data-Variable INPARAM1 wird mit einem Verweis versehen und an das Programm übergeben. Die Parameter werden dem Programm auf die gleiche Weise übergeben wie bei einem Aufruf des Programms von der Befehlszeile aus. Die Parameter, die dem Programm mit dieser Methode übergeben werden, werden als Eingabeparameter behandelt (die dem Programm übergebenen Parameter können vom Programm verwendet und bearbeitet werden, Änderungen an den Parametern werden jedoch nicht zurück an Net.Data übertragen).

Indirekt

Sie übergeben Parameter indirekt, indem Sie *Umgebungsvariablen* verwenden. Umgebungsvariablen sind Zeichenfolgen im Format "name=value", die in einem Umgebungsbereich außerhalb des Programms gespeichert werden. Die Zeichenfolgen werden in einem temporärem Speicherplatz gespeichert, der dem Prozeß zugeordnet wird.

Wenn eine Funktion der Sprachumgebung DTW_SYSTEM aufgerufen wird, werden Eingabeparameter (IN) oder Eingabe-/Ausgabeparameter (INOUT) der Funktion von der Sprachumgebung im Umgebungsbereich gespeichert, bevor die Anweisung im %EXEC-Block ausgeführt wird.

Nach erfolgreicher Ausführung der Anweisung ermittelt die Sprachumgebung DTW_SYSTEM, ob es OUT- oder INOUT-Funktionsparameter gibt. Trifft dies zu, ruft die Sprachumgebung den Wert, der dem Funktionsparameter entspricht, aus dem Umgebungsbereich ab und aktualisiert den Funktionsparameterwert mit dem neuen Wert. Wenn Net.Data die Steuerung erhält, aktualisiert es alle OUT- oder INOUT-Parameter mit den neuen, aus der Sprachumgebung DTW_SYSTEM erhaltenen Werten.

Umgebungsvariablen werden mit den APIs, die in Tabelle 7 beschrieben werden, aktualisiert:

Tabelle 7. Umgebungsvariablen-APIs

ILE-Programmiersprache	Verwenden Sie zum Abrufen...	Verwenden Sie zum Festlegen...
C, C++	getenv()	putenv()
CL(1), RPG, COBOL	QtmhGetEnv()(2)	QtmhPutEnv()(3)

1. Beginnend mit OS/400 V3R7 können Sie auch die CL-Befehle CHGENVVAR und ADDENVVAR verwenden, um eine Umgebungsvariable festzulegen.
2. QtmhGetEnv() ist im Lieferumfang von IBM TCP/IP Connectivity Utilities/400 enthalten.
3. QtmhPutEnv() war ursprünglich nicht im Lieferumfang von IBM TCP/IP ConnectivityUtilities/400 für V3R2 und V3R7 enthalten. Es wurde später hinzugefügt und ist als V3R2-PTF 5763TC1-SF40953 oder V3R7-PTF 5716TC1-SF40954 erhältlich.

Sie können Net.Data-Tabellen an ein Programm übergeben, das von der SYSTEM-Sprachumgebung aufgerufen wird. Das Programm greift auf die Werte eines Net.Data-Makrotabellenparameters anhand der Net.Data-Namen dieser Werte zu. Die Spaltenüberschriften und Feldwerte sind in Variablen enthalten, die mit dem Tabellennamen und der Spaltennummer angegeben werden.

Zum Beispiel sind in der Tabelle myTable die Spaltenüberschriften myTable_N_j und die Feldwerte myTable_N.i_j, wobei i die Zeilennummer und j die Spaltennummer ist. Die Anzahl der Zeilen in der Tabelle ist myTable_ROWS, die Anzahl der Spalten myTable_COLS.

Es ist nicht empfehlenswert, Tabellen mit vielen Zeilen zu übergeben, weil die Anzahl der Umgebungsvariablen für den Prozeß beschränkt ist.

Beispiel für die SYSTEM-Sprachumgebung

Das folgende Beispiel zeigt ein Makro, das die SYSTEM-Sprachumgebung zum Absetzen des Befehls Send Break Message (SNDBRKMSG) an alle Nachrichtenswarteschlangen für Workstations verwendet. Der Text der zu sendenden Nachricht wird aus den Formulardaten (msgToSend) erstellt.

```
%FUNCTION(DTW_SYSTEM) sndbrkmsg () {  
    %EXEC { /QSYS.LIB/SNDBRKMSG.CMD MSG('$ (msgToSend)') TOMSGQ(*ALLWS) %}  
}%  
%HTML(sndbrkmsg) {  
    @sndbrkmsg()  
}%
```

Transaktionsverwaltung mit permanenten Makros

Net.Data bietet Unterstützung für die Transaktionsverarbeitung mit permanenten Makros. Ein *permanentes Makro* ist ein Makro, das integrierte Funktionen enthält, die die Ausführung des Makros als Bestandteil eines permanenten CGI-Prozesses im Web-Server ermöglichen. Das bedeutet, daß mehrere Blöcke eines Makros bzw. mehrere Makros als Bestandteil einer einzelnen logischen Transaktion ausgeführt werden können.

Bei nicht permanenten Makros behandelt Net.Data jeden Makroaufruf als eine vollständige Transaktion. Das bedeutet, daß nach dem Senden einer jeden Antwort an den Browser Datenbanken festgeschrieben, Ressourcen freigegeben und alles wieder in den Ausgangszustand gebracht wird. Der nächste Aufruf des gleichen Makros bewirkt, daß der Status der Anwendung erneut hergestellt wird, und zwar auf der Basis der im Makro als Formulardaten übergebenen Informationen oder der Informationen im Makro selbst. Es gibt keine Möglichkeit, Makrovariablen für mehrere Aufrufe zu speichern, Änderungen an der Datenbank rückgängig zu machen, ohne die gemachten Änderungen explizit aufzuheben, oder Änderungen an der Datenbank in mehreren Browser-Sitzungen als eine vollständige Transaktion zu behandeln.

Mit permanenten Makros können Sie als Anwendungsentwickler Ihre Anwendung auf einer Transaktionsebene erstellen, indem eines oder mehrere Makros in einer permanenten Verbindung aufgerufen werden. Das bedeutet, daß Variablendaten bei mehreren Aufrufen erhalten bleiben, so daß Sie nicht länger Informationen (wie die Anmelde-ID des Benutzers) zwischen mehreren Makroaufrufen als verdeckte Variablen übergeben müssen. Dies gilt auch für Net.Data-Tabellenvariablen, die bei nicht permanenten Makros nicht zwischen mehreren Aufrufen übergeben werden können. Am wichtigsten ist, daß die Anwendung alle Arbeitsschritte rückgängig machen kann, wenn es sich der Benutzer mitten in einer Transaktion anders überlegt.

Anweisungen zum Aufrufen permanenter Makros finden Sie in „Aufrufen eines permanenten Makros“ auf Seite 40.

In diesem Kapitel werden die folgenden Themen beschrieben:

- „Informationen zu permanenten Makros“
- „Definieren einer Transaktion“ auf Seite 124
- „Beispiel für ein permanentes Makro“ auf Seite 132

Informationen zu permanenten Makros

Bei Verwendung von permanenten Makros wird Net.Data in einem speziellen permanenten CGI-Prozeß des Web-Servers ausgeführt, empfängt Eingaben über die Standardeingabe und Umgebungsvariablen und stellt Daten über die Standardausgabe bereit. Nach Rückgabe der Ausgabe an den Web-Server muß dieser den Net.Data-Prozeß jedoch nicht beenden. Statt dessen bleibt der Prozeß aktiv und wartet auf eine Antwort vom Benutzer über den Web-Browser. Weil der Prozeß nicht beendet wird, kann Net.Data Statusinformationen für das Makro beibehalten und Transaktionen geöffnet halten.

Net.Data informiert den Web-Server, daß es in einem permanenten CGI-Prozeß ausgeführt werden will, indem es dem Server neue HTTP-Kopfdaten sendet. Die Unterstützung für die neuen Kopfdaten, „Accept-HTSession“, wurde dem IBM HTTP Server for AS/400 in Version 4, Release 3 (V4R3) hinzugefügt. Net.Data legt fest, welche HTTP-Kopfdaten an den Server gesendet werden, wenn es die erste Ausgabe sendet, weil die Kopfdaten vor der Ausgabe kommen müssen. Das bedeutet, daß Sie bei der Entwicklung eines permanenten Makros die folgenden Punkte berücksichtigen müssen:

- Net.Data muß zum Zeitpunkt der Generierung der ersten Ausgabe aus dem Makro wissen, ob es sich um ein permanentes Makro handelt.
- Sie müssen mit Hilfe der neuen integrierten Funktionen für permanente Makros angeben, daß das Makro permanent ist, bevor Ausgaben generiert werden.

Auf diese Einschränkungen wird in der folgenden Dokumentation hingewiesen.

Die Kenndaten von permanenten Net.Data-Prozessen sind den Kenndaten normaler Net.Data-Prozesse äußerst ähnlich, wobei jedoch die folgenden Ausnahmen gelten:

- Permanente Net.Data-Prozesse werden in einer pseudo-verbindungsorientierten Umgebung ausgeführt. Die Verbindung zwischen Net.Data und dem Web-Server ist permanent, aber zwischen dem Browser und dem Web-Server besteht dennoch keine Verbindung.
- Es sind Transaktionen mit langer Ausführungsdauer möglich. Weil ein einzelner Net.Data-Prozeß mehrere Browser-Anforderungen beinhalten kann, können Transaktionen geöffnet bleiben und je nach nachfolgenden Browser-Anforderungen oder -Fehlerbedingungen festgeschrieben oder rückgängig gemacht werden.
- Ein permanenter Net.Data-Prozeß kann mehr Systemressourcen beanspruchen, weil er potentiell sehr lange Zeit aktiv bleiben kann. Bei der Verwaltung dieser Ressourcen muß sorgfältig vorgegangen werden.
- Die Übertragbarkeit ist eingeschränkt, weil der Web-Server Unterstützung für Permanenz bieten muß.

Definieren einer Transaktion

Eine Transaktion kann einen HTML-Block, mehrere HTML-Blöcke oder mehrere Makros beinhalten. Wenn Sie angeben, daß ein Makro innerhalb einer Transaktion permanent sein soll, müssen Sie Anfang und Ende der Transaktion sowie die HTML-Blöcke, die in die Transaktion einbezogen sind, definieren. Net.Data bietet integrierte Funktionen, mit denen Sie die folgenden Aufgaben für permanente Makros ausführen können:

- „Starten einer Transaktion“ auf Seite 125
- „Angabe der Makro-HTML-Blöcke in einer Transaktion“ auf Seite 126
- „Beenden einer Transaktion“ auf Seite 130
- „Definieren des Gültigkeitsbereichs einer Variablen in einer Transaktion“ auf Seite 130
- „Angabe von COMMIT- und ROLLBACK-Operationen in einer Transaktion“ auf Seite 131

Starten einer Transaktion

Sie starten eine Transaktion, indem Sie Net.Data in Ihrem Makro darüber informieren, daß das Makro permanent ist, bevor Ausgaben an den Browser gesendet werden. Net.Data sendet dann spezielle HTTP-Kopfdaten an den Web-Server, um ihn darüber zu informieren, daß das Makro permanente CGI-Unterstützung benötigt.

Gehen Sie wie folgt vor, um eine Transaktion zu starten:

Verwenden Sie eine der folgenden Methoden im Makro, bevor Ausgaben an den Web-Browser gesendet werden:

- Rufen Sie die integrierte Funktion DTW_STATIC() auf.

Die Funktion DTW_STATIC() informiert Net.Data darüber, daß das aktuelle Makro permanent ist.

Syntax: @DTW_STATIC (["*timeout*"])

Hierbei ist *timeout* ein wahlfreier Parameter, der die Anzahl der Sekunden angibt, die der Web-Server auf eine Antwort vom Browser warten soll, bevor die Transaktion beendet wird.

Beispiel:

```
@DTW_STATIC("60")
%DEFINE {
    var1 = "val1"
    var2 = "val2"
}%
...

%HTML(input){
    ...
}%

%HTML(report){
    ...
}%
```

Für diese Transaktion ist ein Zeitlimit von 60 Sekunden angegeben. Wenn innerhalb von 60 Sekunden keine Antwort vom Browser empfangen wird, beendet der Web-Server die Transaktion. Dies hat keine Auswirkung auf die aktuelle Seite im Browser. Die nächste Seite jedoch, die Bestandteil der Transaktion gewesen wäre, ist nun Bestandteil einer neuen Transaktion.

- Definieren Sie eine Variable mit dem Attribut STATIC.

Syntax: %DEFINE(STATIC) var1 = "val1"

Beispiel:

```
%DEFINE(STATIC) var1 = "val1"
%DEFINE var2 = "val2"
...
%HTML(input){
...
%}
%HTML(report){
...
%}
```

Eine statisch definierte Variable behält ihren Wert im Verlauf einer Transaktion, die mehrere Net.Data-Aufrufe beinhalten kann.

Angeben der Makro-HTML-Blöcke in einer Transaktion

Sie definieren die HTML-Blöcke, die Bestandteil Ihrer Transaktion sind, indem Sie eine Kennung, die *Transaktionskennung*, in der URL-Anforderung verwenden, mit der die HTML-Blöcke aufgerufen werden. Die Definition und Verwendung einer Transaktionskennung umfaßt drei Schritte:

1. Definieren Sie die Transaktionskennung in Ihrem Makro.
2. Rufen Sie die integrierte Funktion DTW_ACCEPT auf, um den Kennungs-namen an Net.Data und den Web-Server zu übergeben.
3. Geben Sie die Kennung in der URL-Anforderung an, um den nächsten HTML-Block aufzurufen.

Gehen Sie wie folgt vor, um eine Transaktionskennung zu definieren:

1. Definieren Sie im DEFINE-Abschnitt eine Variable für die Transaktionskennung.

Beispiel:

```
%DEFINE handle=""
```

2. Generieren Sie wahlfrei eine eindeutige Transaktionskennung, indem Sie die integrierte Funktion DTW_RTVHANDLE() im DEFINE-Abschnitt angeben.

Syntax: @DTW_RTVHANDLE(*handle_name*)

Beispiel:

```
@DTW_STATIC()

%DEFINE handle = ""
@DTW_RTVHANDLE(handle)
```

Die Transaktionskennung kann eine beliebige gültige Zeichenfolge sein. Die Funktion DTW_RTVHANDLE() generiert jedoch aus Sicherheitsgründen eine eindeutige Transaktionskennung, wodurch andere daran gehindert werden, ein Makro aufzurufen, das in Ihrer Transaktion ausgeführt wird.

Gehen Sie wie folgt vor, um eine Transaktionskennung für Net.Data anzugeben:

Geben Sie den Wert der Transaktionskennung für Net.Data mit der integrierten Funktion DTW_ACCEPT() an. Weil diese Kennung Bestandteil der HTTP-Kopfdaten ist, die an den Server gesendet werden, muß die Funktion DTW_ACCEPT() aufgerufen werden, bevor Ausgaben vom Makro generiert werden. Meistens stellt sie das erste Element in Ihrem HTML-Block dar.

Syntax: @DTW_ACCEPT(*handle_name*, [*timeout*])

Hierbei ist *timeout* ein wahlfreier Parameter, der die Anzahl der Sekunden angibt, die der Web-Server auf eine Antwort vom Browser warten soll, bevor die Transaktion beendet wird.

Sie können DTW_ACCEPT() innerhalb oder außerhalb eines HTML-Blocks aufrufen. Wenn die Funktion außerhalb eines HTML-Blocks aufgerufen wird, gelten die Transaktionskennung und das wahlfreie Zeitlimit für alle HTML-Blöcke im Makro.

Beispiel 1: Gibt eine Transaktionskennung an, damit nachfolgende URL-Anforderungen in dieser Transaktion ausgeführt werden

```
@DTW_STATIC()

%DEFINE handle = ""
@DTW_RTVHANDLE(handle)

%HTML(Block1){
@DTW_ACCEPT(handle)
...
%}
```

Wichtig: Wenn Sie DTW_ACCEPT() als erstes Element im HTML-Block aufrufen, müssen Sie sicherstellen, daß es keinen Leerraum zwischen der Zeile, in der die Anweisung %HTML angegeben ist, und dem Aufruf von DTW_ACCEPT() selbst gibt. Net.Data hält den Leerraum für Text, der an den Browser gesendet wird, und löst einen Fehler aus, weil der Aufruf von DTW_ACCEPT() erst gefunden wird, wenn Daten an den Browser gesendet wurden.

Beispiel 2: Gibt eine Transaktionskennung an, die für alle HTML-Blöcke im Makro gilt

```
@DTW_STATIC()

%DEFINE handle = ""
@DTW_RTVHANDLE(handle)
@DTW_ACCEPT(handle)

%HTML(Block1){
...
%}

%HTML(Block2){
...
%}
```

Gehen Sie wie folgt vor, um die Kennung anzugeben, wenn ein HTML-Block aufgerufen wird, der in Ihrer Transaktion ausgeführt werden soll:

Nachdem Sie eine Transaktionskennung generiert und die Funktion DTW_ACCEPT() aufgerufen haben, können nur URL-Adressen mit der betreffenden Transaktionskennung in Ihrer Transaktion ausgeführt werden. Die Transaktionskennung muß in der URL-Adresse direkt nach dem CGI-Programmnamen stehen.

Anmerkung: Wenn Sie die Anweisungen in Ihrem Code eingeben, muß die URL-Adresse im Code in einer Zeile stehen und darf keine Leerzeichen enthalten. Sie wurde hier jedoch zur besseren Lesbarkeit auf zwei Zeilen umbrochen.

- HTML-Programmverbindung (Link):

```
<A HREF="http://server/Net.Data_invocation_path/transaction_handle/
filename/block/[?name=val&...]">any text</A>
```

- HTML-Formular:

```
<FORM METHOD=method
ACTION="http://server/Net.Data_invocation_path/transaction_handle/
filename/block/[?name=val&...]">any text</FORM>
```

- URL-Adresse:

```
http://server/Net.Data_invocation_path/transaction_handle/
filename/block/[?name=val&...]
```

Parameter:

server Gibt den Namen des Web-Servers an. Wenn es sich bei dem Server um den lokalen Server handelt, kann der Server-Name übergangen und eine relative URL-Adresse verwendet werden.

Net.Data_invocation_path

Gibt Pfad und Dateinamen der ausführbaren Net.Data-Datei an. Zum Beispiel /cgi-bin/db2www/.

transaction_handle

Gibt an, welche URL-Adressen Bestandteil einer Transaktion sind, die von einem Net.Data-Makro eingeleitet wird. Diese Kennung wird durch Aufruf der integrierten Funktion DTW_RTVHANDLE ermittelt und muß nach *Net.Data_invocation_path* stehen.

filename

Gibt den Namen der Net.Data-Makrodatei an. Net.Data sucht diesen Dateinamen und versucht, ihn mit den Pfadanweisungen abzugleichen, die in der Initialisierungspfadvariablen MACRO_PATH definiert sind. Weitere Informationen hierzu finden Sie in „MACRO_PATH“ auf Seite 16.

block Gibt den Namen des HTML-Blocks im angegebenen Net.Data-Makro an.

method

Gibt die für das Formular verwendete HTML-Methode an. Hierfür wird METHOD=POST empfohlen.

?name=va/&...

Gibt einen oder mehrere wahlfreie Parameter an, die an Net.Data weitergegeben werden.

Normalerweise stellen Sie HTML-Programmverbindungen (Link) mit diesen URL-Adressen bereit oder geben die URL-Adresse in einem FORM ACTION-Befehl in Ihrem Makro an.

Beispiel 1: Ein typischer HTML-Block mit Programmverbindungen (Links) zu anderen Makroaufrufen, die in der gleichen Transaktion ausgeführt werden

```
@DTW_STATIC()
...
%define handle = ""
@DTW_RTVHANDLE(handle)

%html (report) {
@DTW_ACCEPT(handle)
...
<a href="/cgi-bin/db2www/${handle}/qsys.lib/mylib.lib/
  macros.file/pcgil.mbr/report2">continue</a><br>
<a href="/cgi-bin/db2www/${handle}/qsys.lib/mylib.lib/
  macros.file/pcgil.mbr/quit">quit</a><br>
%}
```

Beispiel 2: Ein typischer HTML-Block mit einer FORM ACTION-Verbindung mit einem anderen Makro

```
@DTW_STATIC()
...
%define handle = ""
@DTW_RTVHANDLE(handle)

%html(input) {
@DTW_ACCEPT(handle)
...
<form method=post action="/cgi-bin/db2www/${handle}/qsys.lib/
mylib.lib/macros.file/pcgil.mbr/report2">
<p>What type of hardware do you want to see?
<menu>
<li><input type="radio" name="hardware" value="MON" checked>Monitors
<li><input type="radio" name="hardware" value="PNT">Pointing devices
<li><input type="radio" name="hardware" value="PRT">Printers
<li><input type="radio" name="hardware" value="SCN">Scanners
</menu>
</form>
%}
```

Beenden einer Transaktion

Sie beenden eine Transaktion, indem Sie Net.Data darüber informieren, daß das Makro nicht länger permanent sein soll.

Gehen Sie wie folgt vor, um die Transaktion zu beenden:

Verwenden Sie die integrierte Funktion DTW_TERMINATE(), um das Ende einer Transaktion anzugeben. Wie die Funktion DTW_ACCEPT() muß auch diese Funktion aufgerufen werden, bevor Ausgaben vom Makro generiert werden, und sie wird normalerweise als erstes Element in einem HTML-Block angegeben.

DTW_TERMINATE informiert Net.Data darüber, daß dieser Aufruf der letzte Aufruf in der aktuellen Transaktion ist.

Syntax: @DTW_TERMINATE()

Diese Funktion unterstützt keine Parameter.

Beispiel:

```
%html(quit) {  
  @DTW_TERMINATE()  
  ...  
%}
```

Definieren des Gültigkeitsbereichs einer Variablen in einer Transaktion

Sie können festlegen, welchen Gültigkeitsbereich eine Variable in einer Transaktion haben soll, indem Sie den Gültigkeitsbereich als Attribut der Anweisung %DEFINE angeben. Sie können folgendes angeben:

Gültigkeitsbereich einer Transaktion

Der Gültigkeitsbereich der Variablen ist die gesamte Transaktion.

Gültigkeitsbereich eines einzelnen Aufrufs

Der Gültigkeitsbereich der Variablen ist ein einziger Net.Data-Aufruf.

Gehen Sie wie folgt vor, um den Gültigkeitsbereich einer Transaktion für eine Variable festzulegen:

Geben Sie das Attribut STATIC an, um festzulegen, daß die Variable den Gültigkeitsbereich einer Transaktion hat, d. h. daß der Wert der Variablen für mehrere Aufrufe in einer Transaktion gespeichert wird. STATIC ist der Standardwert für permanente Makros. Beispiel:

```
@dtw_static()  
%define(static) var1 = "val1"
```

Gehen Sie wie folgt vor, um den Gültigkeitsbereich eines einzelnen Aufrufs für eine Variable festzulegen:

Geben Sie das Attribut TRANSIENT an, um festzulegen, daß die Variable den Gültigkeitsbereich eines einzelnen Aufrufs hat, d. h. daß der Wert der Variablen bei jedem Aufruf erneut initialisiert wird. TRANSIENT ist der Standardwert für nicht permanente Makros. Beispiel:

```
@dtw_static()  
%define(transient) var1 = "val1"
```

In einem permanenten Makro gilt folgendes:

- Alle Variablen, die auf den Aufruf von DTW_STATIC() *folgen*, sind STATIC, wenn sie nicht explizit als TRANSIENT definiert werden.
- Alle Variablen, die dem Aufruf von DTW_STATIC() *vorangehen*, sind TRANSIENT, sofern sie nicht explizit als STATIC definiert werden.

Angeben von COMMIT- und ROLLBACK-Operationen in einer Transaktion

In einem nicht permanenten Makro wird eine COMMIT- oder ROLLBACK-Operation von Net.Data implizit am Ende des Makroaufrufs ausgeführt, je nach Erfolg oder Mißerfolg des Aufrufs. Bei permanenten Makros erfolgt die COMMIT- oder ROLLBACK-Operation nun am Ende der Transaktion. Weil eine Transaktion jedoch viele Makroaufrufe beinhalten kann, können Sie Änderungen innerhalb der Transaktion inkrementell festschreiben oder rückgängig machen.

Gehen Sie wie folgt vor, um anstehende Änderungen in einer Transaktion festzuschreiben:

Geben Sie die integrierte Funktion DTW_COMMIT() an.

Diese Funktion unterstützt keine Parameter und führt alle in der Transaktion anstehenden Änderungen aus.

Beispiel:

```
%html (report) {  
  @dtw_accept(handle)  
  ...  
  %IF (action="Enter")  
    @dtw_commit()  
  %ENDIF  
  
%}
```

Gehen Sie wie folgt vor, um anstehende Änderungen in der Transaktion rückgängig zu machen:

Geben Sie die integrierte Funktion DTW_ROLLBACK() an.

Diese Funktion unterstützt keine Parameter und macht alle in der Transaktion anstehenden Änderungen rückgängig.

Beispiel:

```
%html (report) {  
  @dtw_accept(handle)  
  ...  
  %IF (action="Cancel")  
    @dtw_rollback()  
  %ENDIF  
  
%}
```

Beispiel für ein permanentes Makro

Das folgende einfache Makro enthält mehrere HTML-Blöcke, die in einer einzigen Transaktion ausgeführt werden:

```
@dtw_static()
%define a = "0"
%define(transient) b = "0"
%define handle = ""
@dtw_rtvhandle(handle)

%html (report) {
@dtw_accept(handle)
a = $(a)<br>
b = $(b)<br>
@dtw_add(a, "2", a)
@dtw_add(b, "2", b)
<a href="/cgi-bin/db2www/$(handle)/qsys.lib/mylib.lib/macros.file/pcgil.mbr/report2">
click here to continue</a><br>
<a href="/cgi-bin/db2www/$(handle)/qsys.lib/mylib.lib/macros.file/pcgil.mbr/quit">
click here to quit</a><br>
%}

%html(report2) {
@dtw_accept(handle)
a = $(a)<br>
b = $(b)<br>
@dtw_add(a, "2", a)
@dtw_add(b, "2", b)
<a href="/cgi-bin/db2www/$(handle)/qsys.lib/mylib.lib/macros.file/pcgil.mbr/report3">
Click here to continue</a><br>
<a href="/cgi-bin/db2www/$(handle)/qsys.lib/mylib.lib/macros.file/pcgil.mbr/quit">
Click here to quit</a><br>
%}

%html(report3) {
@dtw_accept(handle)
a = $(a)<br>
b = $(b)<br>
@dtw_add(a, "2", a)
@dtw_add(b, "2", b)
<a href="/cgi-bin/db2www/$(handle)/qsys.lib/mylib.lib/macros.file/pcgil.mbr/quit">
Click here to quit</a><br>
%}

%html(quit) {
@dtw_terminate()
a = $(a)<br>
b = $(b)<br>
done
%}
```


Angenommen, der erste Aufruf bezieht sich auf den HTML-Block `report`. In diesem Fall führt `Net.Data` folgende Schritte aus:

1. Die Funktion `DTW_STATIC()` wird aufgerufen. Sie gibt an, daß dieses Makro permanent ist.
2. Die Variable `a` wird als `STATIC`-Variable erstellt, weil die Standardeinstellung für permanente Makros `STATIC` ist.
3. Die Variable `b` wird als `TRANSIENT`-Variable erstellt, weil sie explizit mit dem Attribut `TRANSIENT` definiert ist.
4. Es wird `DTW_RTVHANDLE()` aufgerufen, wodurch eine Transaktionskennung generiert und in die Variable `handle` gestellt wird.
5. Es wird mit der Verarbeitung des HTML-Blocks `report` begonnen und `DTW_ACCEPT()` aufgerufen, wodurch `Net.Data` darüber informiert wird, daß die Transaktionskennung für diese Transaktion gilt.
6. Es werden Ausgaben an den Browser gefunden, was dazu führt, daß `Net.Data` die HTTP-Kopfdaten an den Web-Server sendet, um anzugeben, daß eine Transaktion beginnt.
7. Die HTML-Seite wird angezeigt. Die Variablen `a` und `b` haben beiden den Wert 0.

Nachdem die erste Ausgabeseite an den Browser gesendet wurde, können die Benutzer die Transaktion entweder fortsetzen oder beenden. Wenn sie fortgesetzt wird, ruft der Web-Server folgende URL-Adresse auf:

```
/cgi-bin/db2www/$(handle)/qsys.lib/mylib.lib/macros.file/pcgil.mbr/report2
```

Der Web-Server erkennt die Transaktionskennung als die Kennung, die von `Net.Data` in den HTTP-Kopfdaten angegeben wurde. Er ruft `Net.Data` als permanentes CGI-Programm auf, was bedeutet, daß der Makroaufruf Teil der aktuellen Transaktion ist.

Wenn der HTML-Block `report2` aufgerufen wird, führt `Net.Data` folgende Schritte aus:

1. Die Funktion `DTW_STATIC()` wird aufgerufen. Sie gibt an, daß dieses Makro permanent ist.
2. Es wird erkannt, daß Variable `a` eine `STATIC`-Variable ist, und der aktuelle Wert wird beibehalten, statt die Variable mit 0 erneut zu initialisieren.
3. Es wird erkannt, daß Variable `b` eine `TRANSIENT`-Variable ist, und es wird ein neues Exemplar der Variablen erstellt, das mit 0 initialisiert wird.
4. Es wird `DTW_RTVHANDLE()` aufgerufen, wodurch eine Transaktionskennung generiert und in die Variable `handle` gestellt wird.
5. Es wird mit der Verarbeitung des HTML-Blocks `report2` begonnen und `DTW_ACCEPT()` aufgerufen, wodurch `Net.Data` darüber informiert wird, daß die Transaktionskennung für diese Transaktion gilt.

6. Es werden Ausgaben an den Browser gefunden, was dazu führt, daß Net.Data die HTTP-Kopfdaten an den Server sendet, um anzugeben, daß eine Transaktion fortgeführt wird.
7. Die HTML-Seite wird angezeigt. Variable *a* hat den Wert 2 und Variable *b* den Wert 0. Der Wert der Variablen *a* stammt aus dem vorherigen Aufruf, weil es sich um eine statische Variable handelt. Der Wert der Variablen *b* wird auf 0 zurückgesetzt.

Nachdem die zweite Seite an den Browser gesendet wurde, können die Benutzer die Transaktion entweder fortsetzen oder beenden. Wenn sie beendet wird, ruft der Web-Server folgende URL-Adresse auf:

```
/cgi-bin/db2www/$(handle)/qsys.lib/mylib.lib/macros.file/pcgil.mbr/quit
```

Der Web-Server erkennt die Transaktionskennung als die Kennung, die von Net.Data in den HTTP-Kopfdaten angegeben wurde, und ruft Net.Data als permanentes CGI-Programm auf, was bedeutet, daß der Makroaufruf Teil der aktuellen Transaktion ist.

Wenn der HTML-Block `quit` aufgerufen wird, führt Net.Data folgende Schritte aus:

1. Die Funktion `DTW_STATIC()` wird aufgerufen. Sie gibt an, daß dieses Makro permanent ist.
2. Es wird erkannt, daß Variable *a* eine `STATIC`-Variable ist, und der aktuelle Wert wird beibehalten, statt die Variable mit 0 erneut zu initialisieren.
3. Es wird erkannt, daß Variable *b* eine `TRANSIENT`-Variable ist, und es wird ein neues Exemplar der Variablen erstellt, das mit 0 initialisiert wird.
4. Es wird `DTW_RTVHANDLE()` aufgerufen, wodurch eine Transaktionskennung generiert und in die Variable `handle` gestellt wird.
5. Es wird mit der Verarbeitung des HTML-Blocks `quit` begonnen und `DTW_TERMINATE()` aufgerufen, wodurch Net.Data darüber informiert wird, daß dies der letzte Aufruf in dieser Transaktion ist.
6. Es werden Ausgaben an den Browser gefunden, was dazu führt, daß Net.Data die HTTP-Kopfdaten an den Server sendet, um anzugeben, daß eine Transaktion beendet wird.
7. Die HTML-Seite wird angezeigt. Variable *a* hat den Wert 4 und Variable *b* den Wert 0.
8. Es werden alle Variablen und andere Ressourcen mit dem Gültigkeitsbereich einer Transaktion gelöscht, weil der Aufruf von `DTW_TERMINATE()` ausgeführt wurde.

Optimieren der Leistung

Das Optimieren der Leistung ist ein wichtiger Bestandteil der Systemoptimierung. In diesem Kapitel werden Strategien zum Optimieren der Leistung von Net.Data erörtert. Folgende Themen werden behandelt:

- „Net.Data-Caching von Makros“
- „Optimieren der Sprachumgebungen“

Stellen Sie zudem sicher, daß Ihr Web-Server entsprechend optimiert wurde. Die Leistung Ihres Web-Servers wirkt sich direkt auf die Antwortzeit aus, unabhängig davon, wie schnell Net.Data ein Makro oder eine Direktanforderung verarbeitet.

Net.Data-Caching von Makros

Bei Net.Data für OS/400 ist Makro-Caching standardmäßig aktiviert und wird verwendet, um den Durchsatz zu verbessern und die CPU-Auslastung zu verringern. Bei aktiviertem Makro-Caching werden vorverarbeitete Makros im Hauptspeicher zwischengespeichert, wenn die Makros zum ersten Mal aufgerufen werden. Diese vorverarbeiteten Versionen sind dann für Wiederverwendung verfügbar. Hierdurch entfällt der Aufwand, der mit dem Lesen der Makros aus HFS und ihrer Verarbeitung bei jeder Anforderung verbunden ist. Die zwischengespeicherte Version eines Makros steht einem Requester zur Verfügung, der über Lesezugriff für die Datei mit dem Makro verfügt.

Der durch die vorverarbeitete Version des Makros belegte Speicherbereich ist ungefähr doppelt so groß wie die Makrodatei selbst. Sie können die Größe des Speicherbereichs, der für das Zwischenspeichern von Makros belegt wird, mit der Caching-Konfigurationsvariable steuern. Weitere Informationen zur Verwendung dieser Variablen finden Sie in „DTW_MACRO_CACHE_SIZE: Variable für Makro-Cache-Größe“ auf Seite 9.

Optimieren der Sprachumgebungen

In den folgenden Abschnitten werden Methoden beschrieben, mit denen Sie die Leistung optimieren können, wenn Sie die von Net.Data bereitgestellten Sprachumgebungen verwenden.

- „REXX-Sprachumgebung“
- „SQL-Sprachumgebung“ auf Seite 136
- „SYSTEM-Sprachumgebung“ auf Seite 137

REXX-Sprachumgebung

Beachten Sie die folgenden Hinweise, um die Leistung Ihrer Net.Data-Anwendung zu optimieren:

- Kombinieren Sie Ihre REXX-Programme, wo möglich. Die Verwendung einer geringeren Anzahl von größeren Programmen bietet eine bessere Leistung als die Verwendung einer größeren Anzahl von kleineren Programmen, weil der REXX-Interpreter bei jedem Aufruf einer REXX-Sprachumgebungsfunktion im Makro initialisiert wird.

- Speichern Sie das REXX-Programm in einer externen Datei, statt es inline in das Net.Data-Makro einzufügen.
- Verweisen Sie bei externen REXX-Programmen in der Befehlszeile in der %EXEC-Anweisung auf die globalen Variablen.
- Übergeben Sie Eingabeparameter direkt an ein REXX-Programm, indem Sie globale Net.Data-Variablen definieren und auf die Variablen verweisen. Rufen Sie bei internen REXX-Programmen die globalen Variablen direkt in Ihrer REXX-Quelle auf.

SQL-Sprachumgebung

Informationen zu DB2-Leistungsüberlegungen finden Sie im Handbuch *DB2 for OS/400 SQL Programming*. Diese Veröffentlichung enthält eine Fülle von Informationen, etwa zur effektiven Verwendung von SQL-Indizes, zur Optimierung der Leistung von Verknüpfungsabfragen und zur Optimierung der Leistung bei der Auswahl von Daten aus mehr als zwei Tabellen.

Verwenden Sie die folgenden Verfahren für die SQL-Sprachumgebung, um die Leistung zu optimieren.

- Verringern Sie die Anzahl der Benutzer-IDs, die eine Verbindung zur Datenbank herstellen, um das Wiederherstellen der Verbindung zu der Datenbank zu vermeiden. Die SQL-Sprachumgebung ordnet jeder fernen Verbindung zu einer Datenbank, die hergestellt wird, ein Benutzerprofil und ein Kennwort zu. Wenn die Variablen LOGIN und PASSWORD nicht mit dem Benutzerprofil und Kennwort übereinstimmen, die einer geöffneten Verbindung zugeordnet sind, wird die Verbindung geschlossen und erneut hergestellt, und die Werte von LOGIN und PASSWORD werden der erneut geöffneten Verbindung zugeordnet.
- Verwenden Sie die Net.Data-Variablen START_ROW_NUM und RPT_MAX_ROWS, um die Größe der zurückgegebenen Tabellen zu verringern. Wenn in einer SQL-Anweisung SELECT die Ergebnismenge hunderte von Datensätzen enthält, übertragen Sie eine Untermenge der Ergebnismenge an den Browser, indem Sie START_ROW_NUM wie einen verschiebbaren Cursor und darüber hinaus RPT_MAX_ROWS verwenden, um die Anzahl der zurückgegebenen Datensätze zu beschränken. Sie müssen beachten, daß Net.Data die Abfrage jedes Mal durchführt, da der Status nicht angegeben wird. Sie können jedoch die Net.Data-Unterstützung für permanente Makros verwenden, um die Ergebnismenge in einer Net.Data-Tabelle zu speichern, die bis zum Ende der Transaktion bestehen bleibt. Informationen zu permanenten Net.Data-Makros finden Sie in „Transaktionsverwaltung mit permanenten Makros“ auf Seite 123.
- Ziehen Sie den Aufruf einer gespeicherten Prozedur in Erwägung, die statisches SQL verwendet. Dynamisches SQL wird zur Laufzeit vorbereitet, während statisches SQL bei der Vorkompilierung vorbereitet wird. Die SQL-Sprachumgebung verwendet dynamisches SQL, so daß SQL-Anweisungen zur Programmlaufzeit ausgeführt werden können. Da die Vorbereitung von Anweisungen zusätzliche Verarbeitungszeit erfordert, ist das statische SQL unter Umständen effizienter.

Beachten Sie, daß die SQL-Steuerkomponente ab OS/400 V4R2 einen Cache für vorbereitete Anweisungen hat. Im Cache speichert die SQL-Steuerkomponente Informationen zu vorbereiteten Anweisungen und hält diese Informationen im Systemspeicher. Wenn die gleiche Anweisung erneut ausgeführt wird, sei dies auch durch einen anderen Benutzer und einen anderen Job, kann die Anweisung viel schneller ausgeführt werden. Der systemweite Cache für vorbereitete Anweisungen ist Teil der normalen SQL-Verarbeitung, für dessen Konfiguration und Aktivierung keine Benutzereingriffe erforderlich sind. Der Cache trägt unter Umständen zur Verringerung von Leistungsvorteilen bei, die statisches SQL gegenüber dynamischem SQL hat.

SYSTEM-Sprachumgebung

Übergeben Sie Eingabeparameter direkt an das Programm, das die SYSTEM-Sprachumgebung aufruft, indem Sie globale Net.Data-Variablen definieren und darauf verweisen.

Net.Data für OS/400 hat eine neue Sprachumgebung namens Direktaufruf eingeführt, die eine einfachere und effektivere Schnittstelle für das Aufrufen von Programmen zur Verfügung stellt. Verwenden Sie die SYSTEM-Sprachumgebung zum Absetzen von Befehlen; verwenden Sie die Direktaufrufsprachumgebung zum Aufrufen von Programmen.

Anhang A. Literaturübersicht

Dieser Anhang enthält eine Liste der Dokumente, auf die in diesem Handbuch verwiesen wird.

- „Net.Data Technical Library“
- „Referenzliteratur“

Net.Data Technical Library

Die Net.Data Technical Library auf der Net.Data-Web-Site ist unter folgender Adresse verfügbar:

<http://www.software.ibm.com/data/net.data/library.html>

Dokument	Beschreibung
<ul style="list-style-type: none">• <i>Net.Data Administration and Programming Guide for OS/390</i>• <i>Net.Data Verwaltung und Programmierung für OS/2, Windows NT und UNIX</i>• <i>Net.Data Verwaltung und Programmierung für OS/400</i>	Enthält Informationen zu Konzepten und Tasks für das Installieren, Konfigurieren und Aufrufen von Net.Data. Außerdem wird das Schreiben von Net.Data-Makros, die Verwendung von Net.Data-Leistungsverfahren und Net.Data-Sprachumgebungen, die Verwaltung von Verbindungen und die Verwendung von Net.Data-Protokoll- und Trace-Funktionen für die Fehlerbehebung und Leistungsverbesserung beschrieben.
<i>Net.Data Reference</i>	Beschreibt die Net.Data-Makrosprache, Variablen und integrierte Funktionen.
<i>Net.Data Language Environment Interface Reference</i>	Beschreibt die Net.Data-Sprachumgebungsschnittstelle.
<i>Net.Data Messages and Codes Reference</i>	Listet die Net.Data-Fehlernachrichten und -Rückkehrcodes auf.

Referenzliteratur

Die folgenden Dokumente können bei der Verwendung von Net.Data und zugehöriger Produkte hilfreich sein:

- *DB2 for OS/400 SQL Programming* SC41-5611
- *OS/400 Distributed Database Programming* SC41-5702

Außerdem sind die OS/400 -Dokumentation und Redbooks, einschließlich Büchern zu DB2 unter folgender URL-Adresse verfügbar:

<http://publib.boulder.ibm.com/html/as400/infocenter.html>

Anhang B. Net.Data-Beispielmakro

Diese Beispielmakroanwendung zeigt eine Liste von Mitarbeiternamen an, aus der der Anwendungsbenutzer zusätzliche Informationen zu einem bestimmten Mitarbeiter abrufen kann, indem er den Namen des Mitarbeiters in der Liste auswählt. Das Makro verwendet die SQL-Sprachumgebung, um die Tabelle EMPLOYEE nach den Mitarbeiternamen und Informationen zu einem bestimmten Mitarbeiter abzufragen.

Die Makrodatei verwendet eine Kopffdatei, die den DEFINE-Block für das Makro enthält.

Abb. 9 auf Seite 142 zeigt das Beispielmakro. Abb. 10 auf Seite 144 zeigt die Kopffdatei.

```

%{***** Sample Macro *****)
*   FileName = sqlsaml.d2w
*   Description:
*       This Net.Data macro queries...
*       - The EMPLOYEE table to create a selection list of
*         employees for display at a browser
*       - The EMPLOYEE table to obtain additional information
*         about an individual employee
*
*****%}
%{*****
*   Include for global DEFINES -
*****%}
%INCLUDE "sqlsaml.hti"
%{*****
*   Function: queryDB           Language Environment: SQL
*   Description: Queries the table designated by the variable myTable and
*                 creates a selection list from the result. The value of the variable
*   myTable is specified in the include file sqlsaml.hti.
*****%}
%FUNCTION(DTW_SQL) queryDB() {
    SELECT FIRSTNME FROM $(myTable)
%MESSAGE {
    -204: {<p><b>ERROR -204: Table $(myTable) not found. </b>
          <p>Be sure the correct include file is being used.</b>
          %} : exit
    +default: "WARNING $(RETURN_CODE)" : continue
    -default: "Unexpected ERROR-$(RETURN_CODE)" : exit
%}

%REPORT {
<select name=emp_name>
    %ROW{
<option>$(V1)
    %}
</select>
%}
%}

%{*****
*   Function: fname           Language Environment: SQL
*   Description: Queries the table designated by the variable myTable for
*                 additional information about the employee identified by the
*                 variable emp_name.
*****%}
%FUNCTION(DTW_SQL) fname(){
    SELECT FIRSTNME, PHONENO, JOB FROM $(myTable) WHERE FIRSTNME='$(emp_name)'
%MESSAGE {
    -204: "Error -204: Table not found "
    -104: "Error -104: Syntax error"
    100: "Warning 100: No records" : continue
    +default: "Warning $(RETURN_CODE)" : continue
    -default: "Unexpected SQL error" : exit
%}
%}

```

Abbildung 9 (Teil 1 von 2). Beispielmakro

```

%{*****
*   HTML block: INPUT                               Title: Dynamic Query Selection   *
*                                                                                       *
*   Description: Queries the EMPLOYEE table to create a selection list of           *
*               the employees for display at the browser                           *
*****%}
%HTML(INPUT){
<html>
<head>
<title>Generate Employee Selection List</title>
</head>
<body>
<h3>$(exampleTitle)</h3>
<p>This example queries a table and uses the result to create
a selection list using a <em>%REPORT</em> block.
<hr>
<form method="post" action="report">
@queryDB()<input type="submit" value="Select Employee">
</form>
<hr>
</body>
</html>
%}
%{*****
*   HTML block:   REPORT                               *
*   Description:  Queries the EMPLOYEE table to obtain additional information *
*               about an individual employee                               *
*****%}
%HTML(REPORT){
<html>
<head>
<title>Obtain Employee Information</title>
</head>
<body>
<h3>You selected employee name = $(emp_name)</h3>
<p>Here is the information for that employee:
<PRE>
@fname()
</PRE>
<hr><a href="input">Return to previous page</a>
</body>
</html>
%}

%{      End of Net.Data macro 1 %}

```

Abbildung 9 (Teil 2 von 2). Beispielmakro

```

=====
%{***** Include File *****}
*   FileName = sqlsamp1.hti                               *
*   Description:                                           *
*       This include file provides global DEFINES for the sqlsamp1.d2w *
*       Net.Data macro.                                   *
*****%}
#define {
    emp_name    = ""
    reposition = sign
    exampleTitle = "Sample Macro"
    myTable = "EMPLOYEE"
    DATABASE = "sample"
}%

%{    End of include file  %}

```

Abbildung 10. Kopfdati

Anhang C. Bemerkungen

Die vorliegenden Informationen wurden für Produkte und Services entwickelt, die auf dem deutschen Markt angeboten werden. Möglicherweise bietet IBM die in dieser Dokumentation beschriebenen Produkte, Services oder Funktionen in anderen Ländern nicht an. Informationen über die gegenwärtig im jeweiligen Land verfügbaren Produkte und Services sind beim IBM Ansprechpartner erhältlich. Hinweise auf IBM Lizenzprogramme oder andere IBM Produkte bedeuten nicht, daß nur Programme, Produkte oder Dienstleistungen von IBM verwendet werden können. Anstelle der IBM Produkte, Programme oder Dienstleistungen können auch andere ihnen äquivalente Produkte, Programme oder Dienstleistungen verwendet werden, solange diese keine gewerblichen Schutzrechte der IBM verletzen. Die Verantwortung für den Betrieb von Fremdprodukten, Fremdprogrammen und Fremdservices liegt beim Kunden.

Für in diesem Handbuch beschriebene Erzeugnisse und Verfahren kann es IBM Patente oder Patentanmeldungen geben. Mit der Auslieferung dieses Handbuchs ist keine Lizenzierung dieser Patente verbunden. Lizenzanfragen sind schriftlich an IBM Europe Director of Licensing 92066 Paris La Defense Cedex France zu richten. Anfragen an obige Adresse müssen auf englisch formuliert werden.

Trotz sorgfältiger Bearbeitung können technische Ungenauigkeiten oder Druckfehler in dieser Veröffentlichung nicht ausgeschlossen werden. Die Angaben in diesem Handbuch werden in regelmäßigen Zeitabständen aktualisiert. Die Änderungen werden in Überarbeitungen oder in Technical News Letters (TNLs) bekanntgegeben. IBM kann jederzeit ohne weitere Mitteilung Verbesserungen und/oder Änderungen an den in dieser Veröffentlichung beschriebenen Produkten und/oder Programmen vornehmen.

Verweise in dieser Veröffentlichung auf Web-Sites anderer Anbieter dienen lediglich als Benutzerinformationen und stellen keinerlei Billigung des Inhalts dieser Web-Sites dar. Das über diese Web-Sites verfügbare Material ist nicht Bestandteil des Materials für dieses IBM Produkt. Die Verwendung dieser Web-Sites geschieht auf eigene Verantwortung.

Lizenznehmer des Programms, die Informationen zu diesem Produkt wünschen mit der Zielsetzung: (i) den Austausch von Informationen zwischen unabhängigen, erstellten Programmen und anderen Programmen (einschließlich des vorliegenden Programms) sowie (ii) die gemeinsame Nutzung der ausgetauschten Informationen zu ermöglichen, wenden sich an folgende Adresse:

IBM Corporation
W92/H3
555 Bailey Avenue
P.O. Box 49023
San Jose, CA 95161-9023
U.S.A.
EADDRESS.

Die Bereitstellung dieser Informationen kann unter Umständen von bestimmten Bedingungen - in einigen Fällen auch von der Zahlung einer Gebühr - abhängig sein.

Die Lieferung des im Handbuch aufgeführten Lizenzprogramms sowie des zugehörigen Lizenzmaterials erfolgt im Rahmen der Allgemeinen Geschäftsbedingungen der IBM oder einer äquivalenten Vereinbarung.

Informationen über Produkte anderer Hersteller als IBM wurden von den Herstellern dieser Produkte zur Verfügung gestellt, bzw. aus von ihnen veröffentlichten Ankündigungen oder anderen öffentlich zugänglichen Quellen entnommen. IBM hat diese Produkte nicht getestet und übernimmt im Hinblick auf

Produkte anderer Hersteller keine Verantwortung für einwandfreie Funktion, Kompatibilität oder andere Ansprüche. Fragen hinsichtlich des Leistungsspektrums von Produkten anderer Hersteller als IBM sind an den jeweiligen Hersteller des Produkts zu richten.

Die oben genannten Erklärungen bezüglich der Produktstrategien und Absichtserklärungen von IBM stellen die gegenwärtige Absicht der IBM dar, unterliegen Änderungen oder können zurückgenommen werden, und repräsentieren nur die Ziele der IBM.

Diese Veröffentlichung dient nur zu Planungszwecken. Die in dieser Veröffentlichung enthaltenen Informationen können geändert werden, bevor die beschriebenen Produkte verfügbar sind.

Diese Veröffentlichung enthält Beispiele für Daten und Berichte des alltäglichen Geschäftsablaufes. Sie sollen nur die Funktionen des Lizenzprogrammes illustrieren; sie können Namen von Personen, Firmen, Marken oder Produkten enthalten. Alle diese Namen sind frei erfunden; Ähnlichkeiten mit tatsächlichen Namen und Adressen sind rein zufällig.

Änderungen in der IBM Terminologie

Die ständige Weiterentwicklung der deutschen Sprache nimmt auch Einfluß auf die IBM Terminologie. Durch die daraus resultierende Umstellung der IBM Terminologie, kann es u. U. vorkommen, daß in diesem Handbuch sowohl alte als auch neue Termini gleichbedeutend verwendet werden. Dies ist der Fall, wenn auf ältere existierende Handbuchausschnitte und/oder Programmteile zurückgegriffen wird.

Zur Vervollständigung ist auch der zugrunde liegende englische Term mit angeführt. Die Änderungen der Termini beziehen sich nicht nur auf die angeführten Einzelwörter, sondern auch auf alle Wortkombinationen (Komposita).

In der nachfolgenden Tabelle finden Sie die betroffenen Termini.

Englischer Terminus	Bisheriger deutscher Terminus	Neuer deutscher Terminus
adaptive	angepaßt	adaptiv
authentication	Identifikationsüberprüfung	Authentifizierung
code page	Zeichenumsetztabelle	Code Page
controller	Steuereinheit	Controller
debugger	Testhilfeprogramm	Debugger
debug/debugging	Testhilfe	(Ausnahme: to debug = mit dem Debugger testen, Fehler beheben)
dynamic link library	Bibliothek für dynamisches Verbinden	Dynamic Link Library
dynamic load library	Bibliothek für dynamisches Laden	Dynamic Load Library
exit	Ausgang	Exit
exit program	Benutzerausgangsprogramm	Exit-Programm
hop	Zwischenschritt	Hop
migrate	umstellen	migrieren
overlay	Überlagerung, Formular	Schablone
terminal	nichtprogrammierbare Datenstation	Terminal
tool	Hilfsprogramm	Tool
trace	Ablaufverfolgung	Trace
workstation	Datenstation	Workstation

Marken

Folgende Namen sind in gewissen Ländern Marken der IBM Corporation:

AIX	Language Environment
AS/400	MVS/ESA
DB2	Net.Data
DB2 Universal Database	OS/2
DRDA	OS/390
DataJoiner	OS/400
IBM	OpenEdition
IMS	

Folgende Namen sind in gewissen Ländern Marken anderer Unternehmen:

Java und alle auf Java basierenden Marken und Logos sind in gewissen Ländern Marken von Sun Microsystems, Inc.

UNIX ist in gewissen Ländern eine eingetragene Marke, die ausschließlich von der X/Open Company Limited lizenziert wird.

Lotus und Domino Go Webserver sind in gewissen Ländern Marken der Lotus Development Corporation.

Microsoft, Windows, Windows NT und das Windows-Logo sind in gewissen Ländern Marken oder eingetragene Marken der Microsoft Corporation.

Andere Firmen-, Produkt- und Servicenamen, die durch zwei Sterne (**) gekennzeichnet sein können, können Marken oder Dienstleistungsmarken anderer Unternehmen sein.

Glossar

Absoluter Pfad (Absolute Path). Der vollständige Pfadname eines Objekts. Absolute Pfadnamen beginnen auf der höchsten Ebene, dem Stammverzeichnis (dieses wird mit einem Schrägstrich (/) oder einem umgekehrten Schrägstrich (\) angegeben).

Aktuelles Arbeitsverzeichnis (Current Working Directory). Das Standardverzeichnis eines Prozesses, ab dem alle relativen Pfadnamen aufgelöst werden.

Anschluß (Port). Eine 16-Bit-Zahl, die für die Kommunikation zwischen TCP/IP und einem Protokoll oder einer Anwendung einer höheren Ebene verwendet wird.

Anwendungsprogrammierschnittstelle (API - Application Programming Interface). Eine vom Betriebssystem oder einem separat erhältlichen Lizenzprogramm zur Verfügung gestellte Funktionsschnittstelle, über die ein in einer höheren Programmiersprache geschriebenes Anwendungsprogramm bestimmte Daten oder Funktionen des Betriebssystems oder des Lizenzprogramms verwenden kann. Net.Data unterstützt die folgenden eigenen Web-Server-APIs zur Leistungsoptimierung über CGI-Prozesse: ICAPI und GWAPI.

API. Anwendungsprogrammierschnittstelle. Net.Data unterstützt drei Web-Server-APIs für eine höhere Leistung über CGI-Prozesse.

Applet. Ein Java-Programm, das in einer HTML-Seite enthalten ist. Applets können mit einem Java-fähigen Browser, wie zum Beispiel Netscape Navigator, verwendet werden und werden geladen, wenn die HTML-Seite verarbeitet wird.

Arbeitseinheit (Unit of Work). Eine wiederherstellbare Operationsfolge, die als eine autarke Operation behandelt wird. Alle Operationen in der Arbeitseinheit können festgeschrieben (COMMIT) oder rückgängig gemacht (ROLLBACK) werden, als wäre nur eine einzelne Operation betroffen. Es können nur Operationen mit Ressourcen, die der COMMIT-Steuerung unterliegen, festgeschrieben oder rückgängig gemacht werden.

BLOB. Großes Binärobjekt (Binary Large Object)

CGI. Common Gateway Interface

CLOB. Großes Zeichenobjekt (Character Large Object)

COMMIT-Steuerung (Commitment Control). Die Festlegung einer Grenze innerhalb des Prozesses, in

dem Net.Data ausgeführt wird, an der Operationen mit Ressourcen Bestandteil einer Arbeitseinheit sind.

Common Gateway Interface (CGI). Ein Standardverfahren zur Übergabe der Steuerung durch einen Web-Server an ein Anwendungsprogramm und zum anschließenden Datenempfang.

DATALINK. Ein DB2-Datentyp, der logische Verweise von der Datenbank auf eine außerhalb der Datenbank gespeicherte Datei aktiviert.

Datenbank. Eine Gruppe von Tabellen oder von Tabellen- und Indexbereichen.

Datenbankverwaltungssystem (DBMS - Database Management System). Ein Softwaresystem, das die Erstellung, den Aufbau und die Änderung einer Datenbank sowie den Zugriff auf die hierin gespeicherten Daten steuert.

Datentyp (Data Type). Ein Attribut aus Spalten und Literalen.

DBCLOB. Großes Objekt aus Doppelbytezeichen

DBMS. Datenbankverwaltungssystem (Database Management System)

Domino Go Webserver. Der von der Lotus Corp. und IBM vertriebene Web-Server, der reguläre sowie sichere Verbindungen bietet. ICAPI und GWAPI sind Schnittstellen für diesen Server.

Firewall. Ein Computer mit Software, die ein internes Netzwerk vor unbefugtem externen Zugriff schützt.

Flat File Interface (FFI). Eine Gruppe von integrierten Net.Data-Funktionen, über die Sie Daten in Dateien mit unverschlüsseltem Text lesen/schreiben können.

GWAPI. Go Web-Server-API

HTML. Hypertext Markup Language

HTTP. Hypertext Transfer Protocol

Hypertext Markup Language. Eine Befehlssprache zum Schreiben von Web-Dokumenten.

Hypertext Transfer Protocol. Ein Übertragungsprotokoll, das zwischen Web-Server und Browser eingesetzt wird.

ICAPI. Internet Connection API. *Siehe 149.*

Internet. Ein internationales, öffentliches TCP/IP-Computernetzwerk.

Intranet. Ein TCP/IP-Netzwerk innerhalb einer firmen-internen Firewall.

Java. Eine vom Betriebssystem unabhängige, objektorientierte Programmiersprache, die sich besonders für Internet-Anwendungen eignet.

LOB. Großes Objekt (Large Object)

Middleware. Software, die zwischen einem Anwendungsprogramm und einem Netzwerk vermittelt. Sie verwaltet die Interaktion zwischen einem Client-Anwendungsprogramm und einem Server über das Netzwerk.

Null. Ein Sonderwert, der angibt, daß keine Informationen vorhanden sind.

Perl. Eine Interpreter-Programmiersprache.

Permanenz (Persistence). Der Zustand, daß ein zugeordneter Wert eine ganze Transaktion lang beibehalten wird. Dabei umfaßt die Transaktion mehrere Net.Data-Aufrufe. Nur Variablen können permanent sein. Außerdem werden Operationen mit Ressourcen, die der COMMIT-Steuerung unterliegen, aktiv gehalten, bis eine explizite COMMIT- oder ROLLBACK-Operation ausgeführt wird oder die Transaktion beendet ist.

Pfadname (Path Name). Informiert das System darüber, wie ein Objekt gefunden werden kann. Der Pfadname wird als Folge von Verzeichnisnamen, gefolgt vom Namen des Objekts, ausgedrückt. Die einzelnen Verzeichnisse und der Objektname werden durch einen Schrägstrich (/) oder einen umgekehrten Schrägstrich (\) getrennt.

Pfad (Path). Eine Suchroute zum Lokalisieren von Dateien.

Registrierdatenbank (Registry). Ein Repository zum Speichern und Abrufen von Zeichenfolgen.

Relativer Pfadname (Relative Path Name).

Ein Pfadname, der nicht auf der höchsten Ebene, dem Stammverzeichnis, beginnt. Das System nimmt an, daß der Pfadname im aktuellen Arbeitsverzeichnis des Prozesses beginnt.

Sprachumgebung (Language Environment).

Ein Modul, das Zugriff von einem Net.Data-Makro auf eine externe Datenquelle wie DB2 oder eine Programmiersprache wie Perl bietet.

TCP/IP. Transmission Control Protocol / Internet Protocol

Transaktion (Transaction). Ein Net.Data-Aufruf. Wenn permanente Variablen verwendet werden, kann eine Transaktion mehrere Net.Data-Aufrufe beinhalten.

Transmission Control Protocol / Internet Protocol.

Eine Gruppe von Übertragungsprotokollen, die Peer-zu-Peer-Konnektivitätsfunktionen sowohl für lokale als auch für Weitverkehrsnetzwerke unterstützen.

URL. Uniform Resource Locator

URL-Adresse (URL - Uniform Resource Locator).

Eine Adresse, die einen HTTP-Server und wahlfrei ein Verzeichnis und einen Dateinamen angibt, wie zum Beispiel:

<http://www.software.ibm.com/data/net.data/index.html>

Web-Server (Web Server). Ein Computer, auf dem eine HTTP-Server-Software wie Internet Connection ausgeführt wird.

Index

A

- Abschnitte eines Makros
 - Darstellung 43
 - Deklaration 43
- Allgemeine Fehler beim Übergeben von Parametern 96, 111
- Allgemeine Funktionen 74
- Arten von Variablen 57
- Aufrufen 99, 112, 113, 120
 - Funktionen 72
 - gespeicherte Prozeduren 112, 113
 - Java-Anwendungen 98
 - Programme, Direktaufruf 93
 - Programme, System 120
 - REXX-Programme 99
 - Sprachumgebungen 92
- Aufrufen von Net.Data 37
 - CGI 37
 - Formulare 37, 41, 128
 - HTML-Blöcke 77
 - mit einem Makro 37
 - Programmverbindungen (Links) 37, 41, 128
 - Übersicht 37
 - URL-Adressen 37, 41, 128
- Ausführbare Variablen 58
- Ausführen von Befehlen 120
- Ausführen von SQL-Anweisungen 102
- Ausgabe, Inaktivieren für Standardberichte 80
- Authentifizierung, Sicherheit 29

B

- Bedingt
 - Logik, IF-Blöcke 86
- Bedingung
 - Variablen 57
- Beispielmakro 139
- Bemerkungen 145
- Benutzerdefinierte Funktionen 66
- Benutzerdefinierte Sprachumgebungen, Anweisungen
 - ENVIRONMENT 7
- Berechtigung
 - Net.Data-Dateien, Angeben von Zugriffsrechten 25
 - Sicherheit 31
- Berichte
 - generieren mehrerer mit einem Funktionsaufruf 82
 - Standard 82
- Berichtsformate anpassen 81
- Berichtsvariablen 64
- BLOBs 105

Blocks, Makro 46

C

- CGI-BIN-Bibliothek, Net.Data-Programmdateiobjekt
 - kopieren 5
- CLOBs 105
- Codieren von DataLink-URL-Adressen in Ergebnismengen 109

D

- DATALINK, Datentyp 109
 - Codieren von URL-Adressen 109
 - DataLink File Manager 109
- Dateien, Angeben von Zugriffsrechten für Net.Data 25
- Datentypen 105, 109, 113
 - DATALINK 109
 - für Direktaufruf 94
 - für gespeicherte Prozeduren 113
 - LOBs 105
- DBCLOBs 105
- DEFINE-Block
 - Beschreibung 46
 - Definieren von Variablen 53
- Definieren von Variablen
 - Abfragezeichenfolgedaten 54
 - DEFINE-Anweisung bzw. -Block 53
 - HTML-Formularbefehle SELECT, INPUT und TEXTAREA 54
- Deklarationsabschnitt, Makrostruktur 43
- Direktaufrufsprachumgebung
 - allgemeine Fehler beim Übergeben von Parametern 96
 - Aufrufen von Programmen 93
 - Übergeben von Parametern 94
 - Übersicht 93
 - unterstützte Datentypen 94
 - Zurückgeben von Werten aus Programmen 97
- DTW_DEFAULT_REPORT 82
- DTW_DIRECTCALL 93
- DTW_JAVA_CLASSPATH 20
- DTW_JAVAPPS 98
- DTW_MACRO_CACHE_SIZE 9
- DTW_PAD_PGM_PARMS 10
- DTW_REXX 99
- DTW_SHOWSQL 10
- DTW_SMTP_CCSD 11
- DTW_SMTP_CHARSET 11
- DTW_SMTP_SERVER 12
- DTW_SQL 102

DTW_SQL_ISOLATION 12
DTW_SQL_NAMING_MODE 14
DTW_SYSTEM 120
DTWR_CLOSE_REGISTRIES 14
Dynamisches Generieren von Variablenamen 55

E

ENVIRONMENT, Anweisungen
 Beispiel 21
 Beschreibung 20
 für benutzerdefinierte Sprachumgebungen 7
 konfigurieren in der Initialisierungsdatei 20, 21
 Parameterliste 21
 Serviceprogramm 21
 Sprachumgebungsart 21
 Syntax 21
Ergebnismengen 115, 116
 mehrere 116
 Richtlinien und Einschränkungen 85
 Standardberichte 116
 nur eine 115
 verarbeiten, gespeicherte Prozeduren 115
Erstellen einer Initialisierungsdatei 8

F

Fehlerbedingungen, Sprachumgebungen 93
FFI_PATH 19
Firewalls 27
Formatieren der Datenausgabe 79
Formulare 37, 39
 Aufrufen von Net.Data 37, 41, 128
 in Web-Seiten zum Aufrufen von Net.Data 39
FUNCTION-Block
 Aufrufen von Funktionen 72
 Beschreibung 47
 Formatieren der Ausgabe 79
 Geltungsbereich für Kennungen 52
Funktionen 112
 allgemeine 74
 aufrufen 72
 Aufrufen gespeicherter Prozeduren 112
 benutzerdefinierte 66
 Beschreibung 66
 definieren 66
 FUNCTION-Blocksyntax 66
 für Tabellen 76
 für unstrukturierte Textdateien 76
 für Web-Registrierdatenbanken 76
 für Wörter 75
 für Zeichenfolgen 75
 MACRO_FUNCTION-Blocksyntax 66
 mathematische 75
 permanente 77

Funktionsaufrufe
 integrierte 72
 Syntax 72
Fußzeilendaten, REPORT-Block 80

G

Geltungsbereich für Kennungen 51
 FUNCTION-Block 52
 global 51
 Makro 52
 REPORT-Block 52
 ROW-Block 52
Gespeicherte Prozeduren 112, 113, 115, 116, 117
 aufrufen vom Makro aus 112
 gültige Datentypen 113
 mehrere Ergebnismengen 116
 mit einer Ergebnismenge 115
 REPORT-Blöcke 116, 117
 Schritte 113
 Standardberichte 115, 116
 Übergeben von Parametern 115
 Verarbeiten von Ergebnismengen 115
Globaler Geltungsbereich für eine Kennung 51
Glossar 149
Große Objekte (LOBs) 105, 107
 Beschreibung 105
 gültige Formate 107

H

HTML 37, 39
 Befehle für Tabellen 80
 Blöcke
 Aufrufen von Net.Data 77
 Beispiel 77
 Beschreibung 48
 Verarbeitung 78
 FORM Submit Button 78
 Formulare 37, 39
 Aufrufen von Net.Data 37, 41, 128
 Informationen zu 39
 SELECT, INPUT und TEXTAREA, Befehle zum
 Definieren von Variablen 54
 generieren in einem Makro 77
 nicht erkannte Daten 78
 Programmverbindungen (Links) 37, 39
 Aufrufen von Net.Data 37, 41, 128
 Informationen zu 39
HTML_PATH 19

I

IF-Blöcke 86
INCLUDE_PATH 18

- Initialisierungsdatei
 - aktualisieren 7
 - Beschreibung 6
 - ENVIRONMENT, Anweisungen 20
 - erstellen 7, 8
 - Format 7
 - Konfigurationsvariablenanweisungen 8
 - Pfadanweisungen 15

J

- Java-Anwendungssprachumgebung
 - Aufrufen von Programmen 98
 - definieren 22
 - Übergeben von Parametern 99
 - Übersicht 98

K

- Konfigurationsvariablenanweisungen
 - Beschreibung 8
 - DTW_MACRO_CACHE_SIZE 9
 - DTW_PAD_PGM_PARMS 10
 - DTW_SHOWSQL 10
 - DTW_SMTP_CCSID 11
 - DTW_SMTP_CHARSET 11
 - DTW_SMTP_SERVER 12
 - DTW_SQL_ISOLATION 12
 - DTW_SQL_NAMING_MODE 14
 - DTWR_CLOSE_REGISTRIES 14
 - konfigurieren in der Initialisierungsdatei 8
- Konfigurieren von Net.Data
 - Definieren der Sprachumgebungen 22
 - Initialisierungsdatei
 - aktualisieren 7
 - Beschreibung 6
 - ENVIRONMENT, Anweisungen 20
 - erstellen 7
 - Konfigurationsvariablenanweisungen 8
 - Pfadanweisungen 15
 - Übersicht 5
 - Zugriffsrechte für Net.Data-Dateien 25
- Kopfdaten, REPORT-Block 80
- Kopieren des Net.Data-Programmdateiobjekts
 - in die CGI-BIN-Bibliothek 5
 - in mehrere Bibliotheken 6

L

- Leistung
 - Optimieren der Sprachumgebungen 135
 - REXX-Sprachumgebung 135
 - SQL-Sprachumgebung 136
 - SYSTEM-Sprachumgebung 137
- Leistungsoptimierung 134

- Listenvariablen 61
- LOBs (große Objekte) 105, 106
 - in SQL- und ODBC-Sprachumgebungen 105
 - unterstützte Typen 106

M

- MACRO_FUNCTION-Block
 - Aufrufen von Funktionen 72
 - Syntax 66
- MACRO_PATH 16
- Makroanforderung 37
 - Beispiele 37
 - Syntax 37
- Makros
 - Aufbau 44
 - bedingte Logik 86
 - Beispiel 44
 - Beschreibung 1
 - Blöcke 46
 - Darstellungsabschnitt 43
 - DEFINE-Block 46
 - Deklarationsabschnitt 43
 - entwickeln 43
 - FUNCTION-Block 47
 - Funktionen 66
 - Geltungsbereich für Kennungen 52
 - Generieren von HTML 77
 - HTML-Block 48
 - IF-Blöcke 86
 - Navigation innerhalb und zwischen 48
 - permanente 123
 - Schleifen 90
 - Variablen 51
 - WHILE-Blöcke 90
- Mathematische Funktionen 75
- Mehrere REPORT-Blöcke 82
- MESSAGE-Block
 - Beispiel 70
 - Beschreibung 70
 - Geltungsbereich 70
 - Syntax 70
 - Verarbeitung 70

N

- Navigation, innerhalb und zwischen Makros 48
- Net.Data
 - aufrufen 37
 - Dateien, Zugriffsrechte 25
 - konfigurieren 5
 - Makros entwickeln 43
 - Sicherheitsmechanismen 31
 - Übersicht 1
- Net.Data-Makro. Siehe Makros. 1

Net.Data-Programmdateiobjekt
in CGI-BIN-Bibliothek kopieren 5
in mehrere Bibliotheken kopieren 6

P

Permanente Funktionen 77
Permanente Makros 123
Pfadanweisungen
Aktualisierungsrichtlinien 15
DTW_JAVA_CLASSPATH 20
EXEC_PATH 17
FFI_PATH 19
HTML_PATH 19
INCLUDE_PATH 18
konfigurieren in der Initialisierungsdatei 15
MACRO_PATH 16
Schützen von Datenbeständen 31
Programmverbindungen (Links) 37, 39
Aufrufen von Net.Data 37, 41, 128
in Web-Seiten zum Aufrufen von Net.Data 39

R

REPORT-Block 116
gespeicherte Prozeduren 116
REPORT-Blöcke 117
Beispiele 82
Beschreibung 79
Einschränkungen 85
Formatieren der Datenausgabe 79
Geltungsbereich 52
gespeicherte Prozeduren 117
Kopf- und Fußzeileninformationen 80
mehrere 82
Richtlinien für mehrere 85
Standardberichte 82
RETURN_CODE, Variable 70, 93
REXX-Sprachumgebung 99, 100
Aufrufen von Programmen 99
Übergeben von Parametern 100
Übersicht 99
ROW-Block, Geltungsbereich für Kennungen 52

S

Schleifen, WHILE-Blöcke 90
Schließungsvariable, Web-Registrierdatenbank 14
Schützen von Datenbeständen 27
Sicherheit
Angaben von Zugriffsrechten 25, 93
Authentifizierung 29
Berechtigung 31
Firewall 27
Net.Data-Mechanismen 31
Netzwerkverschlüsselung 29

Sicherheit (*Forts.*)
Sprachumgebungen 93
Übersicht 27
Sprachumgebungen 99, 120
aufrufen 92
Behandeln von Fehlerbedingungen 93
Beispiele 20
definieren 22
Direktaufruf 93
Java-Anwendung 98
konfigurieren in der Initialisierungsdatei 20
Konfigurieren von Anweisungen
ENVIRONMENT 20
REXX 99
Sicherheit 93
SQL 102
System 120
unterstützte 92
Variablen 65
SQL
Benennungsmoduskonfigurationsvariable 14
Isolationskonfigurationsvariable 12
SQL-Anweisungen ausführen 102
SQL-Sprachumgebung
allgemeine Fehler beim Übergeben von
Parametern 111
Ausführen von SQL-Anweisungen 102
definieren 23
Übersicht 102
SQLCODE-Werte 93
Standardberichte 115, 116
angeben für gespeicherte Prozeduren 115, 116
ausgeben 80
Starten von Net.Data 37
SYSTEM-Sprachumgebung 120
Absetzen von Befehlen 120
Aufrufen von Programmen 120
Übergeben von Parametern 120
Übersicht 120

T

Tabellenfunktionen 76
Tabellenvariablen 62
Tabellenverarbeitungsvariablen 63
Token-Größen 51
Transaktionsverarbeitung 123

U

Übergeben von Parametern 100, 115, 120
Direktaufrufsprachumgebung 94
gespeicherte Prozeduren 115
Java-Anwendungssprachumgebung 99
REXX-Programme 100
SYSTEM-Sprachumgebung 120

Umgebungsvariablen 58
Unstrukturierte Textdateien, Funktionen 76
URL-Adressen 37
 Aufrufen von Net.Data 37, 41, 128
 Definieren von Variablen 54

V

Variablen

 Arten von 51, 57
 ausführbar 58
 Bedingung 57
 Bericht 64
 Beschreibung 51
 definieren 53
 dynamisch generierte Verweise 55
 für Listen 61
 für Tabellen 62
 für Umgebung 58
 Geltungsbereich 51
 Generieren von Namen, dynamisch 55
 Konfiguration, Anweisungen
 Aktivieren von SHOWSQL
 (DTW_SHOWSQL) 10
 Auffüllen von Parametern mit Leerzeichen
 (DTW_PAD_PGM_PARMS) 10
 Beschreibung 8
 E-Mail-SMTP-CCSID (DTW_SMTP_CCSSID) 11
 E-Mail-SMTP-Server (DTW_SMTP_SERVER) 12
 E-Mail-SMTP-Zeichensatz
 (DTW_SMTP_CHARSET) 11
 Inaktivieren von SHOWSQL
 (DTW_SHOWSQL) 10
 Initialisierungsdatei 8
 Makro-Cache-Größe
 (DTW_MACRO_CACHE_SIZE) 9
 Schließen der Web-Registrierdatenbank
 (DTWR_CLOSE_REGISTRIES) 14
 SMTP-Server (DTW_SMTP_SERVER) 12
 SMTP-Zeichensätze
 (DTW_SMTP_CHARSET) 11
 SQL-Benennungsmodus
 (DTW_SQL_NAMING_MODE) 14
 SQL-Isolation (DTW_SQL_ISOLATION) 12
 Sprachumgebung 65
 Tabellenverarbeitung 63
 Token-Größen 51
 verdeckt 60
 verweisen auf 55
 zusätzliche 63
Verarbeiten von Ergebnismengen, gespeicherte Prozeduren 115
Verdeckte Variablen
 Schützen von Datenbeständen 31
 Variablennamen verdecken 60

Verschlüsselung, Netzwerk 29
Verweisen auf Variablen 55

W

Web-Registrierdatenbanken, Funktionen 76
WHILE-Blöcke 90
Wortfunktionen 75

Z

Zeichenfolgefunktionen 75
Zugreifen auf DB2 102
Zugriffsrechte
 für Net.Data-Dateien 25
 für Sprachumgebungen 93
Zurückgeben von Werten aus Programmen 97
Zusätzliche Variablen 63
Zwischenspeichern von Makros, Cache-Größe 9

