



Net.Data

Verwaltung und Programmierung
für OS/400



Net.Data

Verwaltung und Programmierung
für OS/400

Anmerkung

Vor Verwendung dieser Informationen und des darin beschriebenen Produkts sollten die allgemeinen Informationen unter Anhang C, „Bemerkungen“ auf Seite 120 lesen.

Diese Ausgabe gilt für:

- IBM Operating System/400 (Programm 5763-SS1), Version 3 Release 2 Änderung 0
- IBM Operating System/400 (Programm 5716-SS1), Version 3 Release 7 Änderung 0
- IBM TCP/IP Connectivity Utilities for AS/400 (Programm 5763-TC1), Version 3 Release 2 Änderung 0
- IBM TCP/IP Connectivity Utilities for AS/400 (Programm 5716-TC1), Version 3 Release 7 Änderung 0
- IBM HTTP Server for AS/400 (Programm 5769-DG1), Version 4 Release 3 Änderung 0

und alle nachfolgenden Versionen, Releases und Änderungen, sofern in neuen Ausgaben nicht anders angegeben.

Diese Veröffentlichung ist eine Übersetzung des Handbuchs

Net.Data Administration and Programming for OS/400.

herausgegeben von International Business Machines Corporation, USA

© Copyright International Business Machines Corporation 1997, 1998

© Copyright IBM Deutschland Informationssysteme GmbH 1998

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

November 1998

Inhaltsverzeichnis

| | |
|---|-----|
| Vorwort | v |
| Informationen zu Net.Data | v |
| Informationen zu diesem Handbuch | vii |
| Zielgruppe | vii |
| Informationen zu Beispielen in diesem Handbuch | vii |
| Einführung | 1 |
| Was ist Net.Data? | 2 |
| Gründe zur Verwendung von Net.Data | 3 |
| Konfigurieren von Net.Data | 5 |
| Kopieren des Net.Data-Programmdateiobjekts in Ihre CGI-BIN-Bibliothek | 6 |
| Konfigurieren des Web-Servers | 7 |
| Informationen zur Net.Data-Initialisierungsdatei | 8 |
| Anpassen der Net.Data-Initialisierungsdatei | 9 |
| Erstellen einer Initialisierungsdatei | 9 |
| Konfigurationsvariablenanweisungen | 10 |
| Pfadkonfigurationsanweisungen | 13 |
| Umgebungskonfigurationsanweisungen | 17 |
| Erteilen von Zugriffsrechten für Objekte, auf die von Net.Data zugegriffen wird | 19 |
| Sichern der Datenbestände | 21 |
| Verwenden von Firewalls | 21 |
| Verschlüsseln Ihrer Daten im Netzwerk | 24 |
| Verwenden der Authentifizierung | 24 |
| Verwenden der Berechtigung | 25 |
| Verwenden von Net.Data-Mechanismen | 26 |
| Aufrufen von Net.Data | 27 |
| Aufrufen von Net.Data mit einer Makrodatei (Makroanforderung) | 27 |
| HTML-Programmverbindungen (Links) | 29 |
| HTML-Formulare | 30 |
| Aufrufen eines permanenten Makros | 31 |
| Syntax permanenter Makros | 31 |
| Beispiele | 32 |
| Entwickeln von Net.Data-Makros | 33 |
| Aufbau einer Net.Data-Makrodatei | 34 |
| Der DEFINE-Block | 36 |
| Der FUNCTION-Block | 37 |
| HTML-Blöcke | 38 |
| Net.Data-Makrovariablen | 40 |
| Geltungsbereich von Variablen | 41 |
| Definieren von Variablen | 42 |
| Verweisen auf Variablen | 43 |
| Variablenarten | 44 |
| Net.Data-Funktionen | 54 |
| Definieren von benutzerdefinierten Funktionen | 54 |
| Aufrufen von Funktionen | 60 |
| Aufrufen von integrierten Net.Data-Funktionen | 62 |

| | |
|---|------------|
| Generieren von Web-Seiten in einem Makro | 67 |
| HTML-Blöcke | 67 |
| REPORT-Blöcke | 69 |
| Bedingte Logik und Schleifen in einer Makrodatei | 76 |
| Bedingte Logik | 76 |
| Schleifenkonstrukte | 79 |
| Verwenden der Sprachumgebungen | 80 |
| REXX-Sprachumgebung | 81 |
| Konfigurieren der REXX-Sprachumgebung | 81 |
| Aufrufen von externen REXX-Programmen | 81 |
| Übergeben von Parametern | 82 |
| Verwenden der REXX-Anweisung SAY mit OS/400 V3R2 oder V3R7 | 83 |
| Sicherheit | 83 |
| Optimieren der Leistung | 83 |
| Beispiel für die REXX-Sprachumgebung | 84 |
| SQL-Sprachumgebung | 85 |
| Konfigurieren der SQL-Sprachumgebung | 85 |
| Ausführen der SQL-Sprachumgebung unter COMMIT-Steuerung | 86 |
| Verwalten von Datenbankverbindungen für ferne Datenbanken | 87 |
| Gespeicherte Prozeduren | 88 |
| Einschränkungen der SQL-Sprachumgebung | 93 |
| Sicherheit | 93 |
| Optimieren der Leistung | 93 |
| Beispiel für die SQL-Sprachumgebung | 96 |
| SYSTEM-Sprachumgebung | 98 |
| Konfigurieren der SYSTEM-Sprachumgebung | 98 |
| Übergeben von Parametern | 98 |
| Sicherheit | 100 |
| Optimieren der Leistung | 100 |
| Beispiel für die SYSTEM-Sprachumgebung | 100 |
| Transaktionsverwaltung mit permanenten Makros | 101 |
| Informationen zu permanenten Makros | 101 |
| Definieren einer Transaktion | 102 |
| Starten einer Transaktion | 103 |
| Angaben der Makro-HTML-Blöcke in einer Transaktion | 104 |
| Beenden einer Transaktion | 109 |
| Definieren des Gültigkeitsbereichs einer Variablen in einer Transaktion | 110 |
| Angaben von COMMIT- und ROLLBACK-Operationen in einer Transaktion | 111 |
| Beispiel für ein permanentes Makro | 112 |
| Anhang A. Problemanalyse | 115 |
| Anhang B. Net.Data-Beispielmakro | 117 |
| Anhang C. Bemerkungen | 120 |
| Marken | 122 |
| Glossar | 123 |
| Index | 125 |

Vorwort

Vielen Dank, daß Sie sich für Net.Data Version 2, dem IBM Entwicklungshilfsprogramm zum Erstellen dynamischer Web-Seiten 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.

In Net.Data Version 2 wurde die Leistung merklich gesteigert, und außerdem wurden neue Funktionen aufgenommen, mit denen Sie Ihre Internet-Geschäftslösungen mühelos konzipieren und einsetzen können.

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-Seiten erstellen.

Net.Data ist ein Makroumwandler, der als Middleware auf einem Web-Server ausgeführt wird. Sie können Net.Data-Anwendungsprogramme, sogenannte Makros, schreiben, die von Net.Data interpretiert und für die Erstellung dynamischer Web-Seiten mit angepaßtem Inhalt auf Grundlage der Eingabe vom Benutzer, des aktuellen Status Ihrer Datenbanken, vorhandener Geschäftslogik und anderer Faktoren, die Sie in den Makroentwurf aufnehmen, verwendet werden.

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 unterstützt dem Industriestandard entsprechende Schnittstellen wie HTTP (HyperText Transfer Protocol) und CGI (Common Gateway Interface). HTTP wird zwischen dem Browser und dem Web-Server und CGI zwischen dem Web-Server und Net.Data verwendet. Diese Unterstützung ermöglicht Ihnen die Auswahl Ihres bevorzugten Browsers bzw. Web-Servers zur Verwendung mit Net.Data. Die Net.Data-Produktfamilie bietet für die Betriebssysteme OS/400, OS/390, Windows NT, AIX, OS/2, HP-UX, Sun Solaris und SCO ähnliche Funktionen.

Net.Data Version 2 enthält darüber hinaus eine Reihe von Erweiterungen hinsichtlich der Leistung und Funktionalität.

- Die leistungsbezogenen Erweiterungen sind:
 - Fähigkeit, gespeicherte SQL-Prozeduren über die SQL-Sprachumgebung aufzurufen
 - Fähigkeit, eine oder mehrere Ergebnismengen aus gespeicherten SQL-Prozeduren über die SQL-Sprachumgebung abzurufen
 - Unterstützung von Transaktionen, die mehrere Net.Data-Aufrufe beinhalten, durch Verwendung von permanenten Variablen
- Die Erweiterungen der Makrosprache sind:
 - Fähigkeit, Kommentare an einer beliebigen Stelle zu platzieren
 - Verschachtelte IF-Blöcke
 - WHILE-Blöcke
 - MACRO_FUNCTION-Blöcke
 - Ganzzahlige Vergleiche
 - Mehrere REPORT-Blöcke

Informationen zu diesem Handbuch

In diesem Handbuch werden Verwaltungs- und Programmierungskonzepte für Net.Data sowie das Konfigurieren von Net.Data und seiner Komponenten, sein Sicherheitsplan und seine verbesserte Leistung 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 zugreifen, und werden in den Einsatz 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 die 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. Betriebssystemspezifische Unterschiede, Net.Data-Nachrichten und andere Informationen werden im Handbuch *Net.Data Reference* beschrieben.

Als Grundlage zum Verständnis der in diesem Handbuch erläuterten Konzepte müssen Sie wissen, wie ein Web-Server funktioniert, einfache SQL-Anweisungen verstehen und HTML-Befehle, einschließlich HTML-Formularbefehlen, kennen. Sie sollten sich zudem mit den Informationen im Handbuch *Net.Data Reference* vertraut machen.

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 werden Net.Data und die Vorteile, die es bietet, beschrieben.

- „Was ist Net.Data?“ auf Seite 2
- „Gründe zur Verwendung von Net.Data“ auf Seite 3

Was ist Net.Data?

Mit Hilfe von Net.Data-Makros können Sie Logik, Variablen, Funktionsaufrufe und Hilfsprogramme zur Berichterstellung verwenden. Ein Makro ist eine Textdatei mit Net.Data-Makrosprachkonstrukten, HTML-Befehlen, Javascript-Anweisungen und Anweisungen einer Sprachumgebung wie SQL. Net.Data verarbeitet die Makrodatei, 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 zeigt die Beziehung zwischen Net.Data für OS/400 und dem Web-Server und den unterstützten Daten sowie Programmiersprachenumgebungen.

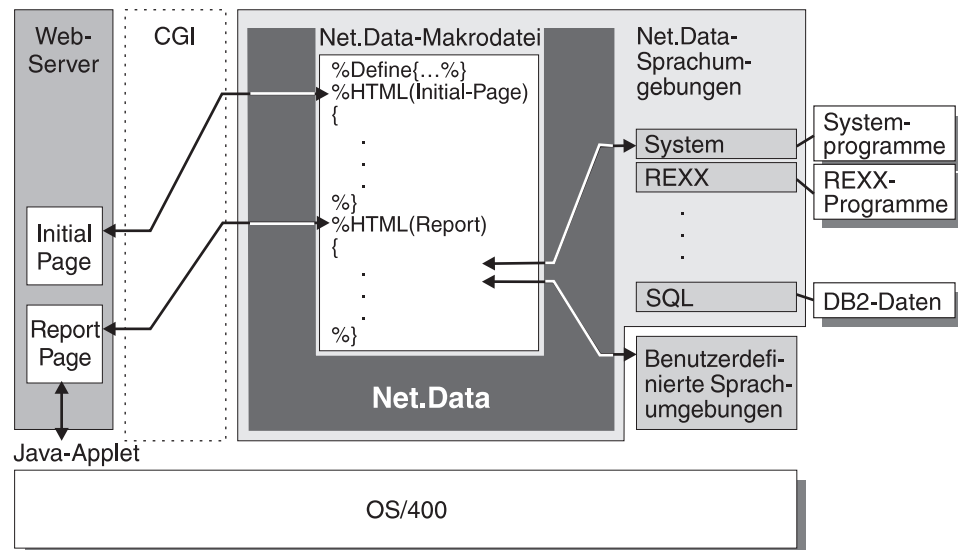


Abbildung 1. Die Beziehung zwischen Net.Data für OS/400, dem Web-Server und unterstützten Daten sowie Programmquellen

Der Web-Server ruft Net.Data als CGI-Anwendung auf, wenn eine URL-Adresse empfangen wird, die Net.Data-Services anfordert. Die URL-Adresse enthält Net.Data-spezifische Informationen, und zwar die zu verarbeitende Makrodatei. 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, wo sie mit dem Browser angezeigt wird.

Gründe zur Verwendung von Net.Data

Net.Data ist eine gute Wahl für das Erstellen dynamischer Web-Seiten, 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 bereit, die auf DB2-Datenbanken zugreifen, oder REXX, Perl und andere Sprachen für Ihre Anwendungen verwenden. Außerdem werden Änderungen an einer Makrodatei sofort in einem Browser sichtbar.

Net.Data erweitert die bestehenden umfangreichen Datenverwaltungsfunktionen von OS/400 um zusätzliche Funktionen zur Unterstützung von kommenden und zukünftigen Formen der Verarbeitung und Darstellung von Internet-Daten. Net.Data bietet für Ihre Anwendungen folgende Vorteile:

- Datengenerierung und -darstellung werden getrennt. Bei Net.Data gibt es keine Einschränkungen hinsichtlich der Darstellung der Daten (z. B. mit HTML oder Javascript). Dadurch können die Benutzer die Darstellung der Daten leicht entsprechend den neuesten Darstellungstrends ändern.
- Vorhandene Fähigkeiten und Geschäftsvorgänge können verwendet werden, um Daten zu generieren, indem Programme, die in C, C++, RPG, COBOL, REXX, Java oder anderen Sprachen geschrieben wurden, eingebunden werden können.
- Internet-Anwendungen können ohne spezielle Programmierkenntnisse entwickelt werden.
- Es wird ein leistungsfähiger Zugriff auf Daten ermöglicht, die in DB2 und in jeder beliebigen DRDA-fähigen Datenbank gespeichert sind.
- 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:

Interpretierte Makrosprache

Die Net.Data-Makrosprache ist eine interpretierte 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 im Abschnitt „Net.Data-Makrovariablen“ auf Seite 40 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 Fehlnachrichten 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

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

Die Software AS/400 TCP/IP und HTTP Server, 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“ auf Seite 6
- „Konfigurieren des Web-Servers“ auf Seite 7
- „Anpassen der Net.Data-Initialisierungsdatei“ auf Seite 9
- „Erstellen einer Initialisierungsdatei“ auf Seite 9
- „Erteilen von Zugriffsrechten für Objekte, auf die von Net.Data zugegriffen wird“ auf Seite 19

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 Create Duplicate Object (CRTDUPOBJ), 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 QHTTSPVR; das Programmdateiobjekt in der Bibliothek QTCP leitet Net.Data-Anforderungen an die Bibliothek QHTTSPVR 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 im Abschnitt „Anpassen der Net.Data-Initialisierungsdatei“ auf Seite 9. Informationen zur Authentifizierung finden Sie im Abschnitt „Verwenden der Authentifizierung“ auf Seite 24.

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.

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. Fügen Sie dazu Anweisungen in die HTTP-Konfigurationsdatei ein, 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/*
```

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.

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 werden von Net.Data nicht verwendet. Wenn Sie Ihre URL-Adresse vereinfachen möchten, können Sie die Anweisung MACRO_PATH in einer Net.Data-Initialisierungsdatei (siehe dazu den folgenden Abschnitt) verwenden.

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 der Konfigurationsvariablen steuern verschiedene Aspekte der Net.Data-Operation.

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 Kopffdateien 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 im Abschnitt „Verwenden der Sprachumgebungen“ auf Seite 80). In diesem Fall müssen alle Verweise auf Makros, Kopffdateien und Programme in der Makrodatei 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 10
- „Pfadkonfigurationsanweisungen“ auf Seite 13
- „Umgebungs-konfigurationsanweisungen“ auf Seite 17

Informationen zum Erstellen einer Initialisierungsdatei finden Sie im Abschnitt „Erstellen einer Initialisierungsdatei“.

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

| | |
|---|---|
| 1 DTWR_CLOSE_REGISTRIES YES | • 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(DTW_REXX) /QSYS.LIB/QTCP.LIB/QTMHREXX.SRVPGM () | • Die Zeilen 5 - 7 definieren die ENVIRONMENT-Anweisungen, die verfügbar sind. |
| 6 ENVIRONMENT(DTW_SQL) /QSYS.LIB/QTCP.LIB/QTMHSQL.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) | |
| 7 ENVIRONMENT(DTW_SYSTEM) /QSYS.LIB/QTCP.LIB/QTMHSYS.SRVPGM () | |

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.) Die ENVIRONMENT-Anweisungen werden nur benötigt, wenn Sie Sprachumgebungen in Ihren Net.Data-Makros aufrufen wollen. Sie müssen auch keine Pfadkonfigurationsanweisungen angeben, wenn Sie bei allen Verweisen auf Dateien innerhalb der Makrodatei den vollständig qualifizierten Pfad angeben.

In den folgenden Abschnitten wird beschrieben, wie Sie die Initialisierungsdatei erstellen und die Konfigurationsanweisungen in der INI-Datei 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 Create Source Physical File (CRTSRCPF), 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 Source Entry Utility (SEU) oder einen Datenstationseditor, 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 im Abschnitt „Erteilen von Zugriffsrechten für Objekte, auf die von Net.Data zugegriffen wird“ auf Seite 19.

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 in Erstellung befindlichen 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, die von den in Net.Data enthaltenen Sprachumgebungen verwendet werden, die Sie in der Initialisierungsdatei angeben können, beschrieben.

- „DTW_SQL_ISOLATION: DB2-Isolationsvariable“ auf Seite 11
- „DTW_SQL_NAMING_MODE: SQL-Tabellenbenennungsvariable“ auf Seite 12
- „DTWR_CLOSE_REGISTRIES: Variable zum Öffnen der Web-Register“ auf Seite 12

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*

Hierbei ist *locking_method* 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 eine relationale Datenbank im relationalen Datenbankverzeichnis angegeben ist und es sich bei der relationalen Datenbank nicht um ein OS/400-System handelt.

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*

Hierbei ist *mode* einer der folgenden Werte:

SQL_NAMING Gibt an, daß Tabellen im folgenden Format durch den Datensammelungsnamen 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 Standarddatensammelungsname nicht angegeben wird. SQL_NAMING ist der Standardtabellenname.

SYS_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 Standarddatensammelungsname (library) angegeben wird.

DTWR_CLOSE_REGISTRIES: Variable zum Öffnen der Web-Register

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

Syntax:

DTWR_CLOSE_REGISTRIES YES*NO

Dabei gilt folgendes:

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

NO Gibt an, daß alle offenen Web-Register 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 ein Web-Register (mit den integrierten Funktionen des Web-Registers) zu erhöhen, indem Register weniger oft geöffnet und geschlossen werden müssen.

Pfadkonfigurationsanweisungen

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

- „MACRO_PATH“
- „EXEC_PATH“ auf Seite 14
- „INCLUDE_PATH“ auf Seite 15
- „FFI_PATH“ auf Seite 16

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

Aktualisierungsrichtlinien:

Mehrere allgemeine Richtlinien gelten für alle Pfadanweisungen.

- Jedes angegebene Verzeichnis wird durch ein Semikolon (;) begrenzt.
- 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.

Hinweis: Net.Data durchsucht alle angegebenen Verzeichnisse, aber nicht die Unterverzeichnisse. Wenn sich zum Beispiel Net.Data-Makros in den folgenden Verzeichnissen befinden, müssen Sie jedes Unterverzeichnis in der Pfadanweisung angeben:

```
/usr/test/client  
/usr/test/assoc  
/usr/test/partner
```

Ihre Anweisung MACRO_PATH könnte in etwa so aussehen:

```
MACRO_PATH [=] /usr/test/client;usr/test/assoc;usr/test/partner
```

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-Makrodateien 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;...;path_n

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 im Abschnitt „Aufrufen von Net.Data“ auf Seite 27.

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 mindestens ein Verzeichnis an, das von Net.Data nach einem externen Programm durchsucht wird, 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;...;path_n

Beispiel: Das folgende Beispiel zeigt die Anweisung EXEC_PATH in der Initialisierungsdatei und die Anweisung EXEC in der Makrodatei, die 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/rexxprgms.file gefunden wird, ist der qualifizierte Name des Programms /qsys.lib/rexx.lib/rexxprgms.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 mindestens ein Verzeichnis an, das 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 Kopfddatei an die Pfadangabe an, um den qualifizierten Kopfddateinamen 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 Kopfddatei 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 Kopfddatei /u/user1/includes/myInclude.txt.

Beispiel 2: Das folgende Beispiel zeigt die Anweisung INCLUDE_PATH und eine Kopfddatei, die durch einen Unterverzeichnisnamen vollständig qualifiziert ist.

Net.Data-Initialisierungsdatei:

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

Net.Data-Makro:

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

Die Kopfddatei 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 Kopfddatei /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 mindestens ein Verzeichnis an, in dem 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*.

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 im Abschnitt „Verwenden der Sprachumgebungen“ auf Seite 80 und zusammen mit der Schnittstelle für Sprachumgebungen auch im Handbuch *Net.Data Language Environment Reference* beschrieben.

Für den Aufruf der Sprachumgebung erfordert Net.Data, daß in der Net.Data-Initialisierungsdatei eine Anweisung ENVIRONMENT für eine Sprachumgebung vorhanden ist.

Net.Data gibt mehrere Variablen an, die sich darauf auswirken, wie Net.Data-Sprachumgebungen Aufrufe an in FUNCTION-Blöcken definierte Funktionen interpretieren. Die Einstellungen dieser Variablen müssen an eine Sprachumgebung übergeben werden, um wirksam zu werden.

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-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-Umgebungskonfiguration 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. Unter OS/400 wird der Name des Serviceprogramms 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.

Die Parameter werden nach den in der FUNCTION-Blockdefinition angegebenen Parametern in das Feld *parm_data_array* der Struktur *dtw_lei* übergeben. (Informationen zu diesen Strukturen finden Sie im Handbuch *Net.Data Language Environment Reference*.)

Sie müssen diese Parameter als Konfigurationsvariablen oder als Variablen in Ihrer Makrodatei 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 INI-Datei 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.

```

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/QTMHSQL.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/QTMHSYS.SRVPGM ( )
```

Jede Anweisung ENVIRONMENT muß in einer separaten Zeile stehen.

Erteilen von Zugriffsrechten für Objekte, auf die von Net.Data zugegriffen wird

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 Net.Data-Programmdateien und -Serviceprogramme und Durchsuchen der Verzeichnisse (Bibliotheken) in den Pfaden zu den Programmdateien und Serviceprogrammen
- Lesen der entsprechenden Net.Data-Makrodateien und Durchsuchen der entsprechenden, durch die Pfadkonfigurationsanweisung `MACRO_PATH` angegebenen Verzeichnisse
- Ausführen der entsprechenden Dateien und Durchsuchen der entsprechenden, durch die Pfadkonfigurationsanweisung `EXEC_PATH` angegebenen Verzeichnisse
- Lesen der entsprechenden Dateien und Durchsuchen der entsprechenden, durch die Pfadkonfigurationsanweisung `INCLUDE_PATH` angegebenen Verzeichnisse
- Lesen und Schreiben der entsprechenden Dateien und Durchsuchen der entsprechenden, durch die Pfadkonfigurationsanweisung `FFI_PATH` angegebenen Verzeichnisse

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 QTMHHTP1 eine Berechtigung erteilt (in V3R2 und V3R7 hat Internet Connection für AS/400 CGI-Programme nur unter dem Benutzerprofil QTMHHTP1 ausgeführt). :

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

```
CHGAUT OBJ('/WWW') USER(QTMHHTP1) DTAAUT(*RX)
CHGAUT OBJ('/WWW/macro') USER(QTMHHTP1) DTAAUT(*RX)
CHGAUT OBJ('/WWW/macro/*') USER(QTMHHTP1) DTAAUT(*RX)
```

Sie müssen allen Objekten im Pfad eine Berechtigung erteilen.

- Verwenden Sie im Bibliotheksdateisystem (QSYS.LIB) den CL-Befehl Grant Object Authority (GRTOBJAUT), 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 80 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 24
- „Verwenden der Authentifizierung“ auf Seite 24
- „Verwenden der Berechtigung“ auf Seite 25
- „Verwenden von Net.Data-Mechanismen“ auf Seite 26

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 des 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 einen Firewall konfiguriert haben, mit dem 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 wird. Die Firewall-Software wird für die Erstellung eines ersten Firewall zwischen Web-Server und öffentlichem Internet sowie einem zweiten Firewall zwischen Web-Server und gesichertem Intranet, in dem sich der DB2-Server befindet, verwendet. Abb. 3 zeigt diese Konfiguration.

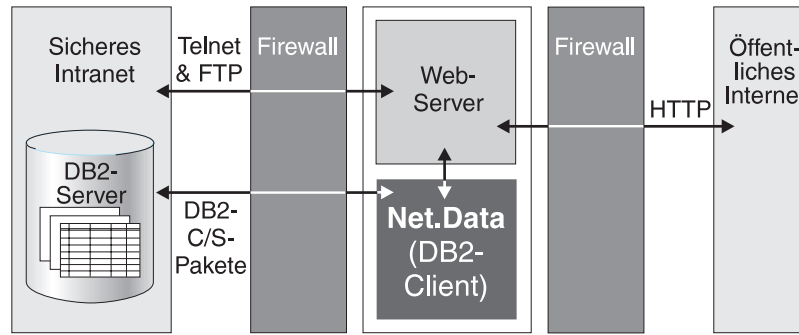


Abbildung 3. Konfiguration mit hoher Sicherheit

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 Intranet auf den DB2-Server zugreifen kann. Konfigurieren Sie dafür den Firewall so, daß ein DB2-Datenverkehr durch den Firewall möglich ist. Eine Möglichkeit besteht darin, Regeln zur Paketfilterung hinzuzufügen, die Anforderungen von DB2-Clients 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 Internet 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 des Firewall auf einer Datenstationsplattform. Diese Konfiguration ist einfacher als die oben beschriebene, bietet jedoch trotzdem einen Datenbankschutz. Abb. 4 zeigt diese Konfiguration.

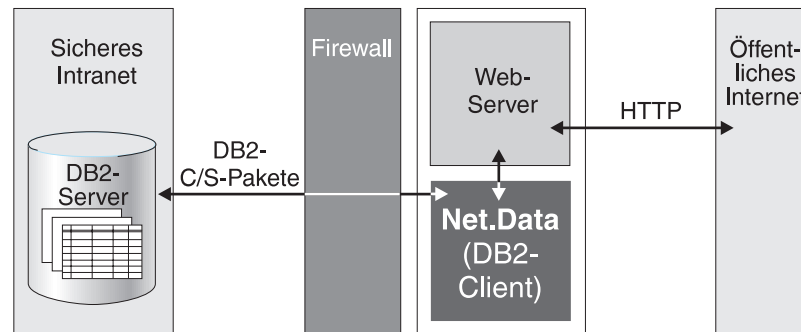


Abbildung 4. Konfiguration mit mittlerer Sicherheit

Der Firewall muß so konfiguriert sein, daß Anforderungen von DB2-Clients 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 des Firewall und des gesicherten Intranet installiert. Dadurch sind sie nicht gegen externe Manipulation geschützt. Für diese Konfiguration benötigt der Firewall keine Regeln zur Paketfilterung. Abb. 5 zeigt diese Konfiguration.

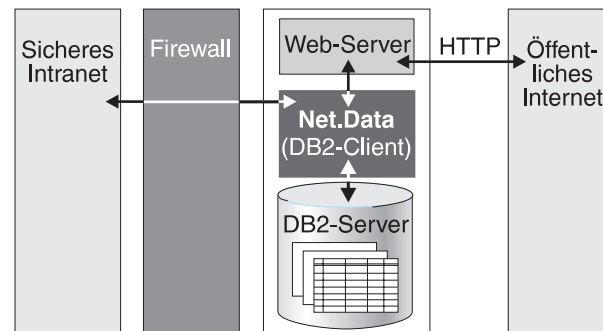


Abbildung 5. Konfiguration mit niedriger Sicherheit

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-abhängig 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 mit V4R1 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-Makrodateien 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 im Abschnitt „Erteilen von Zugriffsrechten für Objekte, auf die von Net.Data zugegriffen wird“ auf Seite 19.

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 im Abschnitt „Verwenden der Authentifizierung“ auf Seite 24. 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

Außer den oben beschriebenen Methoden können Sie andere von Net.Data bereitgestellte Mechanismen, wie Pfadanweisungen und verdeckte Variablen verwenden, sowie Methoden, die HTML-Formulare oder SQL-Anweisungen verwenden.

Pfadanweisungen

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

Verdeckte Variablen

Mit verdeckten 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 hierzu finden Sie im Abschnitt „Verdeckte Variablen“ auf Seite 48.

Sie können auch die folgenden Methoden für die Erstellung eines Schutzplans verwenden:

HTML-Formulare

Erstellen Sie mit Hilfe von Net.Data Ihren eigenen Schutzplan. Beispielsweise könnten Sie anhand eines HTML-Formulars Informationen zur Gültigkeitsprüfung von einem Benutzer anfordern und diese Informationen dann anhand von Daten aus einer Datenbank oder durch ein externes Programm, das von einem Net.Data-Makro aufgerufen wird, überprüfen lassen.

SQL-Anweisungen

Schützen Sie Ihre Datenbestände durch die SQL-Anweisungen, die Benutzer an die Datenbank senden dürfen; begrenzen Sie zum Beispiel SELECT-Anweisungen auf zwei Tabellen.

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 einer Makrodatei 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 im Abschnitt „Transaktionsverwaltung mit permanenten Makros“ auf Seite 101.

Der Name der Net.Data-Makrodatei und der Name des innerhalb des Net.Data-Makros auszuführenden HTML-Blocks werden in der Programmverbindung (Link), im Formular bzw. in der URL-Adresse angegeben.

In diesem Kapitel wird das Aufrufen von Net.Data mit einer Makrodatei beschrieben.

- „Aufrufen von Net.Data mit einer Makrodatei (Makroanforderung)“
- „Aufrufen eines permanenten Makros“ auf Seite 31

Aufrufen von Net.Data mit einer Makrodatei (Makroanforderung)

In diesem Abschnitt wird gezeigt, wie Sie Net.Data durch Angabe einer Makrodatei aufrufen können. Beim Aufruf über eine Makroanforderung können Sie Net.Data mit einer HTML-Programmverbindung (Link), einem HTML-Formular oder über eine URL-Adresse aufrufen.

Die folgenden Beispiele für eine Anweisungssyntax zeigen die unterschiedlichen Möglichkeiten zum Aufrufen von Net.Data.

- HTML-Programmverbindung (Link):

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

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

```
http://server/Net.Data_invocation_path/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 Net.Data-Programmdatei 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 im Abschnitt „MACRO_PATH“ auf Seite 13.

block Gibt den Namen des HTML-Blocks in der angegebenen Net.Data-Makrodatei an.

method

Gibt die für das Formular verwendete HTML-Methode an. Hierfür wird METHOD=POST empfohlen.

?name=val&...

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

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://MyServer/cgi-bin/db2www/myMacro.d2w/report">
.
.
.
</A>
```

Beispiel 2: Aufrufen von Net.Data mit einem Formular

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

Anmerkung: Die URL-Adresse muß im Code in einer Zeile stehen, wurde hier jedoch zur besseren Lesbarkeit auf zwei Zeilen umbrochen.

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

- „HTML-Programmverbindungen (Links)“
- „HTML-Formulare“ auf Seite 30

HTML-Programmverbindungen (Links)

Sie können auf einer Web-Seite eine Programmverbindung (Link) erstellen, die zur Ausführung eines HTML-Blocks führt, indem Sie den HTML-Befehl `<a>` in der Makrodatei verwenden. Sie erzielen dies durch Verwendung des Attributs `HREF` zur Angabe des Makros und des HTML-Blocks und durch Aufnahme von Text oder sogar eines Bilds in den Programmverbindungsbehl. Dadurch wird der Text bzw. das Bild als „Detailpunkt“ angegeben, wenn die Web-Seite im Browser angezeigt wird. Wenn ein Benutzer über einen Browser den Text bzw. das Bild anklickt, führt Net.Data den HTML-Block im Makro aus.

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>
```

Die Programmverbindung (Link) ruft das folgende Makro auf:

```
%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 69.

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 Seite 29 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, eine Makrodatei 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 im Abschnitt „Transaktionsverwaltung mit permanenten Makros“ auf Seite 101.

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 Net.Data-Programmdatei 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 im Abschnitt „MACRO_PATH“ auf Seite 13.

block Gibt den Namen des HTML-Blocks in der angegebenen Net.Data-Makrodatei an.

method

Gibt die für das Formular verwendete HTML-Methode an. Hierfür wird METHOD=POST empfohlen.

?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 einer Makrodatei

```
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. in der Makrodatei 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.

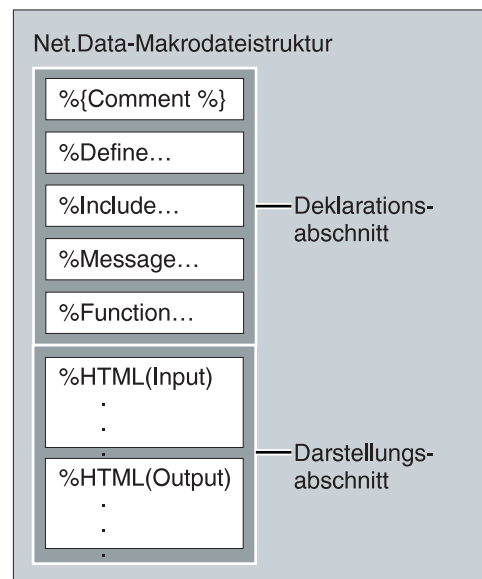


Abbildung 6. Makrodateistruktur

- Der *Deklarationsabschnitt* enthält die Definitionen von Variablen und Funktionen in der Makrodatei.
- 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 Makrodateiabschnitte und der Konstrukte.

Hinweis zu Berechtigungen: Stellen Sie sicher, daß der Web-Server Zugriffsrechte für diese Datei hat. Weitere Informationen hierzu finden Sie im Abschnitt „Erteilen von Zugriffsrechten für Objekte, auf die von Net.Data zugegriffen wird“ auf Seite 19.

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

- „Aufbau einer Net.Data-Makrodatei“
- „Net.Data-Makrovariablen“ auf Seite 40
- „Net.Data-Funktionen“ auf Seite 54
- „Generieren von Web-Seiten in einem Makro“ auf Seite 67
- „Bedingte Logik und Schleifen in einer Makrodatei“ auf Seite 76

Aufbau einer Net.Data-Makrodatei

Die Makrodatei 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 im Abschnitt „Erteilen von Zugriffsrechten für Objekte, auf die von Net.Data zugegriffen wird“ auf Seite 19.

- 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 der Makrodatei an. Die HTML-Blöcke werden im Abschnitt „HTML-Blöcke“ auf Seite 38 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>@rex1(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 erfordert zwei Parameter: den Namen des zu verarbeitenden Makros und den anzuzeigenden HTML-Block in diesem Makro.

Wenn die Makrodatei 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 in der Makrodatei 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, die einen Funktionsaufruf an ein externes REXX-Programm und einen Funktionsaufruf an eine in der Makrodatei enthaltene Funktion definieren.

```
%{ ***** FUNCTION Block *****%}
%FUNCTION(DTW_REXX) rexx1 (IN input) returns(result) { <-- Diese Funktion
                                                         akzeptiert einen Parameter
                                                         und gibt ein Ergebnis zurück,
                                                         das den zugeordneten
                                                         Funktionsaufruf ersetzt.

    %EXEC{ompsamp
.mbr %} <-- Diese Funktion führt ein externes REXX-Programm aus,
                                                         das "ompsamp.mbr
" heißt.
%}

%FUNCTION(DTW_REXX) today () RETURNS(result) {
    result = date() <-- Die einzelne Quellenanweisung für
                                                         diese Funktion befindet sich inline.
%}
```

Der erste FUNCTION-Block, rexx1, ist eine 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 tritt an die Stelle der im OUTPUT-Block enthaltenen aufrufenden Funktion @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 das gesamte REXX-Programm (eine ganze Zeile) selbst innerhalb der Funktionsdeklaration. Ein externes Programm ist nicht erforderlich. Inline-Programme sind für REXX- und Perl-Funktionen zulässig, da REXX und Perl interpretierte 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 in der Beispielmakrodatei heißt INPUT. Der INPUT-Block 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 von 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. Sie können einen beliebigen Namen vergeben. 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 Übergabe des Formulars werden die in dieses Feld eingegebenen Daten in der Variablen *input_data* an den HTML-Block OUTPUT übergeben.

Sie können über einen relativen Verweis auf HTML-Blöcke in anderen Makrodateien zugreifen, wenn sich diese Makrodateien auf demselben Web-Server befinden. So greift beispielsweise der ACTION-Parameter ACTION="../othermacro.d2w/main" auf den HTML-Block main in der Makrodatei othermacro.d2w zu. Auch hier werden alle in das Formular eingegebenen Daten in der Variablen *input_data* an das andere 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>
```

Zum Empfang von Eingabedaten müssen keine Umgebungsvariablen definiert werden, wie es bei den meisten CGI-Programmen der Fall ist. Net.Data verarbeitet Umgebungsvariablen für Sie. Sie müssen die Variablennamen lediglich angeben.

Der nächste HTML-Block des Beispiels ist der Block OUTPUT. Er enthält HTML-Code und Net.Data-Makroanweisungen, die die Verarbeitung der Ausgabe von der INPUT-Blockanforderung 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 INPUT-Block 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 legt die Anzahl und die Arten der übergebenen Variablen fest.

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 zum ersten Mal auf sie verwiesen 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. jeweils aus Name und Wert bestehenden URL-Datenpaaren 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 Variablen“ auf Seite 41
- „Definieren von Variablen“ auf Seite 42
- „Verweisen auf Variablen“ auf Seite 43
- „Variablenarten“ auf Seite 44

Geltungsbereich von Variablen

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 der Makrodatei aus verwiesen werden kann. Kennungen mit globalem Geltungsbereich sind:

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

- Makrodatei

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.) Eine Kennung mit dem Geltungsbereich einer Makrodatei ist von der Stelle, an der sie deklariert wird, bis zum Ende der Makrodatei sichtbar.

- 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. Wenn die Kennung innerhalb eines Funktionsblocks verwendet wird, behält sie den Wert bei, der ihr vor dem Funktionsaufruf zugeordnet wurde. Wird sie innerhalb des Funktionsblocks aktualisiert, behält die Kennung den neuen Wert nach dem Funktionsaufruf bei.

- 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.

Wenn auf eine Variable verwiesen wird, wird sie durch den Wert der Kennung ersetzt. Wenn ein Verweis auf eine Variable keinen zugeordneten Wert hat oder ein Funktionsaufruf keinen Wert zurückgibt, wird der Verweis durch eine leere Zeichenfolge ersetzt.

Definieren von Variablen

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

- **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 der Anweisung ist für Net.Data spezifisch:

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

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

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

Sollen Anführungszeichen in einer Zeichenfolge verwendet werden, setzen Sie jeweils zwei Anführungszeichen hintereinander. Zwei aufeinanderfolgende Anführungszeichen allein entsprechen einer leeren Zeichenfolge. Beispiel:

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

Die Variable HI enthält den Wert say "hello".

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

Die Variable reply enthält den Wert hello.

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

Die Variable empty ist Null.

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 und INPUT**

Sie können die Befehle SELECT und INPUT in einem HTML-Formular verwenden. Im folgenden Beispiel werden Standardbefehle für HTML-Formulare zum Definieren einer Variablen verwendet:

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

oder

```
<SELECT NAME="variable_name">
```

variable_name ist der Name, den Sie der Variablen geben. Der Wert der Variablen wird durch die im Formular empfangene Eingabe bestimmt. Im Abschnitt „HTML-Formulare“ auf Seite 30 finden Sie ein Beispiel dafür, wie diese Art der Variablendefinition in einem Net.Data-Makro verwendet wird.

Ein Variablenwert, der durch einen Befehl INPUT oder SELECT empfangen wird, überschreibt einen mit einer Anweisung DEFINE in einem Net.Data-Makro definierten Variablenwert.

- **URL-Daten**

Sie können Net.Data-Makros über URL-Anforderungen aufrufen und Variablen, wie zum Beispiel eine Benutzer-ID, in die URL-Angabe einfügen, die an Net.Data gesendet werden sollen. 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 den Befehl INPUT empfängt. Net.Data empfängt und verarbeitet die Daten ebenso 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.

Wenn Sie Variablen als Teil Ihrer Textdarstellungsanweisungen verwenden wollen, fügen Sie Verweise auf Variablen in Ihre HTML-Blöcke ein. 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 Verbindung (Link) erstellen:

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

Verweise auf Variablen können in allen Abschnitten eines Net.Data-Makros eingefügt werden. Wenn die Variable zu dem Zeitpunkt, zu dem auf sie verwiesen wird, noch nicht definiert ist, gibt Net.Data eine leere Zeichenfolge zurück. Net.Data definiert die Variable nicht.

Einschränkung: Rückbezügliche Verweise (bzw. Zyklen) sind nicht zulässig. Die folgenden DEFINE-Anweisungen verursachen beispielsweise einen Fehler, wenn auf die Variable verwiesen wird und die endgültigen Werte berechnet werden:

```
%DEFINE a="$(b)"  
%DEFINE b="$(a)"
```

Variablenarten

Sie können die folgenden Arten von Variablenverweisen in Ihren Makrodateien verwenden:

- „Bedingungsvariablen“ auf Seite 45
- „Umgebungsvariablen“ auf Seite 45
- „Ausführbare Variablen“ auf Seite 46
- „Verdeckte Variablen“ auf Seite 48
- „Listenvariablen“ auf Seite 49
- „Tabellenvariablen“ auf Seite 50
- „Zusätzliche Variablen“ auf Seite 51
- „Variablen zur Tabellenverarbeitung“ auf Seite 51
- „Berichtsvariablen“ auf Seite 52
- „Sprachumgebungsvariablen“ auf Seite 53

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 für die einzelnen Variablen 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
```

Im Abschnitt „Listenvariablen“ auf Seite 49 finden Sie ein Beispiel für die Verwendung von Bedingungsvariablen mit Listenvariablen.

Umgebungsvariablen

Sie können auf Net.Data-Umgebungsvariablen verweisen, die in dem Prozeß existieren, unter dem Net.Data ausgeführt wird.

Die Syntax zur Definition von Umgebungsvariablen sieht folgendermaßen aus:

```
%define var=%ENVVAR
```

Dabei ist *var* der Name der Variable, 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
```

Im Handbuch *Net.Data Reference* finden Sie weitere Informationen zur Anweisung ENVVAR.

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 zu Sprachkonstrukten 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 einer Makrodatei 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 unter „EXEC_PATH“ auf Seite 14.
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 Feldgruppe identischer Datensätze oder Zeilen sowie eine Feldgruppe von Spaltennamen, die die Felder in jeder Zeile beschreiben. Eine Tabelle wird in einem Net.Data-Makro wie in der folgenden Anweisung definiert:

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

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

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

Bei der Definition einer Tabelle hat die Tabelle null Zeilen und null Spalten, und es wird kein Speicher zugeordnet. 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 allen anderen Sprachumgebungen, wie zum Beispiel DTW_REXX oder DTW_PERL, werden Tabellenwerte nicht automatisch gesetzt. Statt dessen werden die einzelnen Elemente der Tabelle für den nativen Sprach-Interpreter als Ausgabe-parameter zur Verfügung gestellt und müssen dann durch die Prozedur bzw. das Programm, das ausgeführt wird, gesetzt werden. Weitere Informationen dazu, wie Tabellenvariablen von Sprachumgebungen verwendet werden, finden Sie im Abschnitt „Verwenden der Sprachumgebungen“ auf Seite 80.

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 verwiesen werden. Nähere Informationen hierzu finden Sie unter „Variablen zur Tabellenverarbeitung“ auf Seite 51. 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.

Die Spaltennamen und Feldwerte in einer Tabelle werden als Feldgruppenelemente mit dem Ursprung 1 und nicht der Standardkonvention der Sprachen C und C++ entsprechend mit dem Ursprung 0 für den Anfang von Feldgruppen 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 der Makrodatei 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:

```
<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 der Makrodatei 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 der Makrodatei verwendet. Zum Beispiel können Sie den Namen der aktuellen Spalte, die verarbeitet wird, folgendermaßen abrufen:

```
<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 die Makrodatei 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 Variable 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. Diese Variablen müssen vor ihrer Verwendung in einem Variablenverweis mit Hilfe einer Anweisung DEFINE oder der Funktion @DTW_ASSIGN() definiert werden. Setzen Sie Sprachumgebungsvariablen, oder verweisen Sie auf sie in den entsprechenden Net.Data-Makroblöcken.

Mit diesen Variablen können Sie Tasks durchführen wie das Herstellen von Verbindungen zu Datenbanken und das Aktivieren der Sprachenunterstützung.

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

Sie können zwei Arten von Funktionen in Net.Data-Makros verwenden:

Benutzerdefinierte Funktionen (UDFs - user-defined functions)

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

Integrierte Net.Data-Funktionen

Die Funktionen, die von Net.Data zur Verwendung in Ihrer Anwendung bereitgestellt werden, etwa Funktionen zur Wort- und Zeichenfolgenbearbeitung und Funktionen, mit denen Tabellenvariablenfunktionen abgerufen und festgelegt werden.

In diesem Abschnitt werden die folgenden Themen beschrieben:

- „Definieren von benutzerdefinierten Funktionen“
- „Aufrufen von Funktionen“ auf Seite 60
- „Aufrufen von integrierten Net.Data-Funktionen“ auf Seite 62

Definieren von benutzerdefinierten Funktionen

Zum Definieren eigener Funktionen in der Makrodatei verwenden Sie einen FUNCTION-Block oder einen MACRO_FUNCTION-Block.

FUNCTION-Block

Definiert eine Unterroutine, die von einem Net.Data-Makro aufgerufen wird und entweder von der Sprachumgebung verarbeitet wird oder ein externes Programm aufruft.

MACRO_FUNCTION-Block

Definiert eine Unterroutine, die von einem Net.Data-Makro aufgerufen wird und von Net.Data, d. h. nicht von einer anderen Sprachumgebung, verarbeitet werden muß. Die Anweisungen im Block müssen Quellenanweisungen der Net.Data-Makrosprache sein.

Der MACRO_FUNCTION-Block ist eine Alternative zum FUNCTION-Block, die die Leistung verbessern kann. Der MACRO_FUNCTION-Block wird nur von Net.Data verarbeitet und ruft keine Sprachumgebung auf. Einzelheiten zu diesen beiden Konstrukten finden Sie im Handbuch *Net.Data Reference*.

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

FUNCTION-Block:

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

MACRO_FUNCTION-Block:

```
%MACRO_FUNCTION function-name(usage parameter, ...) {  
    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.

Mit Net.Data werden verschiedene Standardsprachumgebungen bereitgestellt.

function-name

Gibt den Namen des FUNCTION- oder MACRO_FUNCTION-Blocks an. Sie können den FUNCTION-Block oder MACRO_FUNCTION-Block über einen Funktionsaufruf an einer anderen Stelle in der Makrodatei ausführen. Der Funktionsaufruf verweist auf den *function-name* mit einem vorangestellten kommerziellen A (@). Nähere Informationen finden Sie unter „Aufrufen von Funktionen“ auf Seite 60.

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 eine Funktion aufruft, werden alle FUNCTION-Blöcke gleichen Namens bzw. MACRO_FUNCTION-Blöcke gleichen Namens 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 einen FUNCTION-Block bzw. einen MACRO_FUNCTION-Block übergeben, von ihm empfangen oder sowohl übergeben als auch empfangen wird. Die Verwendungsart gilt für alle nachfolgenden Parameter in der Parameterliste, bis sie durch eine andere Verwendungsart geändert wird. Die Standardart ist IN.

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 des FUNCTION-Blocks bzw. MACRO_FUNCTION-Blocks ungültig.

return-var

Geben sie diesen Parameter nach dem Schlüsselwort RETURNS an, um einen speziellen OUT-Parameter zu definieren. Der Wert der Rückkehrvariablen wird dem Funktionsaufruf zugeordnet und ersetzt den Funktionsaufruf bei der Net.Data-Makroverarbeitung. 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
- Der 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. Vor der Übergabe der ausführbaren Anweisungen an die Sprachumgebung ersetzt Net.Data diese Variablenverweise bzw. Funktionsaufrufe durch die tatsächlichen Werte.

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 im Abschnitt „Ausführbare Variablen“ auf Seite 46.

Bei MACRO_FUNCTION-Blöcken sind die ausführbaren Anweisungen eine Kombination aus HTML-Anweisungen und Net.Data-Makrosprachkonstrukten. In diesem Fall spielt die Sprachumgebung keine Rolle, da Net.Data als Sprachprozessor fungiert und die ausführbaren Anweisungen auswertet und ausführt.

report-block

Definiert einen oder mehrere REPORT-Blöcke zur Behandlung der Ausgabe des FUNCTION-Blocks. Siehe „REPORT-Blöcke“ auf Seite 69 .

message-block

Definiert den MESSAGE-Block, der alle vom FUNCTION-Block zurückgemeldeten Nachrichten verarbeitet. Siehe „MESSAGE-Blöcke“ auf Seite 58.

Definieren Sie Funktionen in der äußersten Net.Data-Makroebene, bevor diese im Net.Data-Makro aufgerufen werden.

Verwenden von Sonderzeichen in Funktionen

Wenn Zeichen, die der Syntax der Net.Data-Sprachkonstrukte entsprechen, im Abschnitt 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 bynumber 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. Net.Data verwendet den folgenden MESSAGE-Blockprozeß:

1. Net.Data setzt die Variable RETURN_CODE, eine Sprachumgebungsvariable, für jeden Funktionsaufruf an einen FUNCTION-Block. RETURN_CODE wird nicht bei einem Funktionsaufruf an einen MACRO_FUNCTION-Block gesetzt.
2. Wenn eine Sprachumgebung einen Rückkehrcodewert an Net.Data zurückgibt, setzt Net.Data den Wert von RETURN_CODE auf den Wert des Rückkehrcodes.
3. Nach Beendigung des Funktionsaufrufs bestimmt der MESSAGE-Block anhand des Werts von RETURN_CODE die weitere Vorgehensweise.

Ein MESSAGE-Block besteht aus einer Reihe von Nachrichtenweisungen, 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 FUNCTION-Blocknamen bzw. einem MACRO_FUNCTION-Blocknamen:

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

function_name

Gibt den Namen des FUNCTION-Blocks oder MACRO_FUNCTION-Blocks an, der aufgerufen werden soll. Die Funktion muß bereits im Net.Data-Makro definiert sein, es sei denn, es handelt sich um eine integrierte Funktion.

argument Gibt den Namen einer definierten Variablen, einer Literalzeichenfolge, eines Variablenverweises oder eines Funktionsaufrufs an. Die Argumente in einem Funktionsaufruf werden mit den Parametern eines FUNCTION-Blocks bzw. MACRO_FUNCTION-Blocks abgeglichen, und jedem Parameter wird bei der Verarbeitung des FUNCTION-Blocks bzw. des MACRO_FUNCTION-Blocks der Wert des entsprechenden Arguments zugewiesen. Die Argumente müssen dieselbe Anzahl und Art wie die zugehörigen Parameter aufweisen.

Net.Data verarbeitet den FUNCTION-Block, MACRO_FUNCTION-Block oder integrierte Funktionen, die einem Funktionsaufruf zugeordnet sind, in der folgenden Reihenfolge:

1. Net.Data verarbeitet die Variablenverweise und Funktionsaufrufe im Abschnitt mit den ausführbaren Anweisungen des FUNCTION-Blocks. Net.Data ersetzt alle Variablenverweise durch die aktuellen Werte der Variablen, führt alle Funktionsaufrufe aus und ersetzt sie durch den Rückkehrwert des jeweiligen Funktionsaufrufs. Die Variablenverweise und Funktionsaufrufe werden in der Reihenfolge verarbeitet, in der sie angegeben sind. Net.Data verarbeitet während dieses Schritts keine integrierten Funktionen oder MACRO_FUNCTION-Blöcke.
2. Der native Sprachprozessor verarbeitet den Abschnitt mit den ausführbaren Anweisungen. Bei FUNCTION-Blöcken entspricht der Prozessor der für den FUNCTION-Block definierten Sprachumgebung wie zum Beispiel SQL, REXX oder Perl. Bei MACRO_FUNCTION-Blöcken fungiert Net.Data als Sprachprozessor und führt die ausführbaren Anweisungen aus. Integrierte Funktionen enthalten keine ausführbaren Anweisungen. Net.Data verarbeitet die integrierten Funktionen nach Funktionsnamen.

Net.Data übergibt die Parameter der Funktion an den nativen Sprachprozessor. Net.Data übergibt nur für IN- und INOUT-Parameter Werte an den nativen Sprachprozessor und akzeptiert vom nativen Sprachprozessor zurückgegebene Werte nur für OUT- und INOUT-Parameter.
3. Net.Data setzt die impliziten Variablen RETURN_CODE und DTW_DEFAULT_MESSAGE entsprechend dem Rückkehrcode und der zurückgegebenen Nachricht des Sprachprozessors. Für MACRO_FUNCTION-Blöcke setzt Net.Data diese Variablen nicht.

4. Wenn der FUNCTION-Block bzw. MACRO_FUNCTION-Block einen oder mehrere REPORT-Blöcke enthält oder angibt, daß Standardberichte generiert werden sollen, verarbeitet Net.Data den Bericht bzw. die Berichte unter Verwendung der neuen Werte aller angegebenen Ausgabeparameter. Für integrierte Funktionen generiert Net.Data keine Berichte.
5. Wenn der FUNCTION-Block einen lokalen MESSAGE-Block enthält, verarbeitet Net.Data den MESSAGE-Block. Net.Data verarbeitet den globalen MESSAGE-Block, wenn eine der folgenden Bedingungen zutrifft:
 - Ein globaler MESSAGE-Block ist angegeben, und der Rückkehrcode wird nicht von einem lokalen MESSAGE-Block bearbeitet.
 - Eine integrierte Funktion wird aufgerufen.

Net.Data verarbeitet keine MESSAGE-Blöcke für MACRO_FUNCTION-Blöcke.

6. Net.Data ersetzt den Funktionsaufruf durch den zurückgegebenen Wert der Funktion. Bei FUNCTION-Blöcken ist dieser Wert einer der folgenden:

RETURNS-Parameterwert Ersetzt den FUNCTION-Block mit einem Schlüsselwort RETURNS.

Leere Zeichenfolge ("") Ersetzt den FUNCTION-Block ohne Schlüsselwort RETURNS, wenn der Wert für RETURN_CODE gleich Null ist.

RETURN_CODE Ersetzt den FUNCTION-Block ohne Schlüsselwort RETURNS, wenn der Wert für RETURN_CODE *ungleich* Null ist.

Bei MACRO_FUNCTION-Blöcken ersetzen die Ergebnisse der Verarbeitung des Abschnitts mit den ausführbaren Anweisungen den Funktionsaufruf.

Bei integrierten Funktionen hängt der Wert vom Format der integrierten Funktion ab.

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, so daß Sie diese Funktionen nicht mehr in einem FUNCTION-Block definieren müssen. Rufen Sie diese Funktionen einfach in einem Makro an einer beliebigen Stelle auf, an der eine benutzerdefinierte Funktion aufgerufen werden kann.

Eine integrierte Funktion wird auf gleiche Weise aufgerufen wie eine benutzerdefinierte Funktion, nämlich mit dem Net.Data-Funktionsaufruf. Abb. 7 zeigt, wie die Net.Data-Funktionen und die Makrodatei miteinander interagieren.

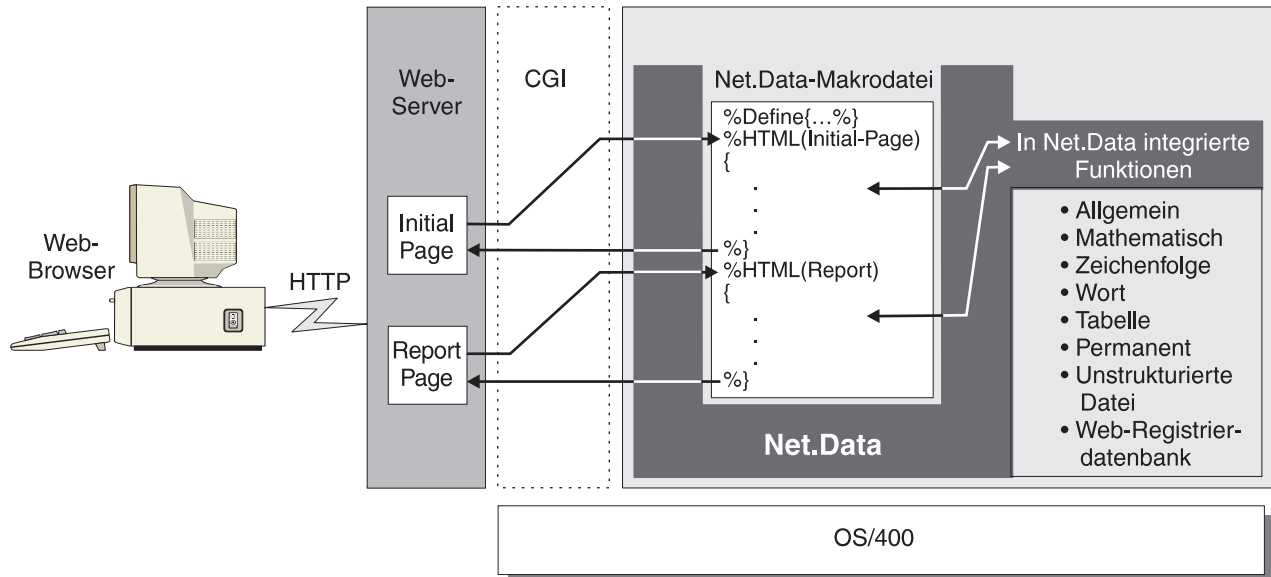


Abbildung 7. In Net.Data integrierte Funktionen

Net.Data verarbeitet innerhalb eines HTML-Blocks einen Funktionsaufruf für die integrierte Funktion. Net.Data verarbeitet die Funktion und gibt dann die Ergebnisse innerhalb eines HTML-Blocks in der Makrodatei zurück.

Es gibt drei Methoden, mit denen integrierte Funktionen ihre Ergebnisse zurückgeben können. An dem zugehörigen Präfix können Sie erkennen, wie eine Funktion ihre Ergebnisse ausgibt:

- **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 für Web-Registrierdatenbankfunktionen.)
- **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 den 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.

- „Allgemeine Funktionen“ auf Seite 64
- „Mathematische Funktionen“ auf Seite 65
- „Zeichenfolgefunktionen“ auf Seite 65
- „Wortfunktionen“ auf Seite 65
- „Tabellenfunktionen“ auf Seite 66
- „Funktionen für unstrukturierte Textdateien“ auf Seite 66
- „Funktionen für Web-Registrierdatenbanken“ auf Seite 66
- „Permanente Funktionen“ auf Seite 67

Allgemeine Funktionen

Mit Hilfe dieser Gruppe von Funktionen können Sie Web-Seiten durch Ändern von Daten oder Zugreifen auf Systemservices entwickeln. Hiermit können Sie Umgebungsvariablen abfragen und setzen, HTML-Escape-Codes verwenden 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 der Makrodatei 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>
%}
```

Mit der Funktion DTW_URLESCSEQ werden Zeichen, die in einer URL-Adresse nicht erlaubt 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 Ergebnisgenauigkeit 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_rPOWER("2", "-3", result)
```

DTW_POWER gibt ".125" 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 Zeichenfolgewert 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 dieselbe 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 einer Zeichenfolge 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. Sie können diese Funktionen auch verwenden, um die Werte in einer Net.Data-Tabellenvariable abzurufen und festzulegen. Tabellenvariablen enthalten einen Wertebereich mit den 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 ein anderer Prozeß weder lesen noch aktualisieren kann. DTWF_CLOSE gibt die Datei frei, wenn Net.Data sie nicht länger benötigt, damit die anderen Prozesse 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.

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

Die Net.Data-Makrodatei enthält HTML-Blöcke und die Konstrukte in den HTML-Blöcken, die Textdarstellungsanweisungen wie HTML für einen Web-Browser generieren. In einer Makrodatei muß mindestens ein HTML-Block definiert werden. Jeder HTML-Block generiert eine einzelne Web-Seite beim Browser. Net.Data verarbeitet bei jedem Aufruf nur einen HTML-Block, und der Inhalt des HTML-Blocks steuert den übrigen Net.Data-Aufruf. Wenn Sie eine Anwendung erstellen möchten, die mehrere Web-Seiten umfaßt, können Sie Net.Data mehrmals aufrufen, um HTML-Blöcke mit normalen HTML-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 Verbindung anklickt, ruft der Web-Browser Net.Data auf, und Net.Data analysiert die Makrodatei. 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 als HTML-Anweisung interpretiert und zum Anzeigen an den Browser gesendet.

Wenn der Benutzer eine Auswahl getroffen und den Knopf zur Übergabe (Submit) gedrückt hat, führt Net.Data den ACTION-Abschnitt des HTML-Elements FORM aus, der einen Aufruf an den Block HTML(REPORT) des Net.Data-Makros definiert. Net.Data verarbeitet daraufhin den Block HTML(REPORT) 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 im `INPUT`-Formular empfangenen Wert ersetzt wurde, führt Net.Data die Abfrage aus. An dieser Stelle beginnt Net.Data wieder damit, den HTML-Bericht an den Browser zu senden, der die Ergebnisse der Abfrage gemäß den im `REPORT`-Block definierten Textdarstellungsanweisungen anzeigt.

Wenn Net.Data die Verarbeitung des `REPORT`-Blocks abgeschlossen hat, kehrt es zum Block `HTML(REPORT)` 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 `HTML`-Befehlen, Makrovariablenverweisen und Funktionsaufrufen angegeben werden kann. Wahlfrei kann im `REPORT`-Block ein Tabellennamen angegeben werden. Wenn kein Tabellennamen angegeben wird, verwendet Net.Data die Tabellendaten aus der ersten Ausgabetabelle in der Parameterliste des `FUNCTION`-Blocks. Wenn Sie keine Tabelle im `FUNCTION`-Block angeben, verwendet Net.Data die Standardtabellendaten.

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

- Kopfdaten mit `HTML`-Daten, die einmal vor den Tabellenzeilendaten angezeigt werden
- `ROW`-Block mit `HTML`- und Tabellenvariablen, die einmal für jede Zeile der Ergebnistabelle angezeigt werden
- Fußzeilen mit Daten, die einmal nach den Tabellenzeilendaten angezeigt werden

Beispiel:

```
%REPORT{
<H2>Query Results</H2>
<P>Select a name for details.
<TABLE BORDER=1>
<TR><TD>Name</TD><TD>Location</TD>
  %ROW{
<TR>
<TD>
<a href="/cgi-bin/db2www/name.mac/details?name=$(V1)
&location;=$(V2)">$(V1)</a></TD>
<TD>$(V2)</TD>
  %}
</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-Makroergbnistabelle. Diese Variablen sind im Abschnitt „Variablen zur Tabellenverarbeitung“ auf Seite 51 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 oder 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 außerdem benutzerdefinierte Funktionen und Variablen im REPORT-Block verwenden.
- Wenn Net.Data einen Standardbericht mit vorformatiertem Text ausgeben soll, geben Sie den REPORT-Block nicht in der Makrodatei an. 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 den Wert 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:

```
%FUNCTION(DTW_SQL) custlist() {  
    SELECT Name, Phone, Fax FROM CustomerTbl  
    %REPORT{  
<I>Phone Query Results:</I>  
<BR>  
=====
```

| |
|--------------------------|
| Name: Doen, David |
| Phone: 422-245-1293 |
| Fax: 422-245-7383 |

```
-----  
<BR>  
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 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 Werte für Name, Phone und Fax werden für jede abgerufene Zeile in die Variablen V1, V2 und V3 eingesetzt. Dieser Text bildet den Fußzeilenbereich des REPORT-Blocks.
3. Nach jeder abgerufenen Zeile wird zur besseren Lesbarkeit eine Linie gezeichnet.
4. Die Zeichenfolge *Total records retrieved:* und der Wert für NUM_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 88). 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 der Funktionsparameterliste übergeben. Wenn Sie mehr Tabellen in der Parameterliste übergeben als Sie REPORT-Blöcke angegeben haben und 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 nur in der SQL-Sprachumgebung der DTW_DEFAULT_REPORT-Wert YES dem Wert MULTIPLE 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: 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.

Beispiel 3: MACRO_FUNCTION-Block

```
%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.

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

    %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 in der Parameterliste des FUNCTION-Blocks erscheinen: zuerst für **table1** und dann für **table2**.

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 Tabellen in der FUNCTION-Parameterliste übergeben werden, werden Standardberichte für die restlichen Tabellen in der Reihenfolge angezeigt, in der sie in der Parameterliste des FUNCTION-Blocks erscheinen: 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:**

- Geben Sie einen REPORT-Block pro Tabelle an. Der für den REPORT-Block angegebene Parameter für den Tabellennamen muß mit dem entsprechenden Parameter für den 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.
- Wenn die Standardverarbeitung erfolgen soll, 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 Angabe 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, die von Net.Data für OS/400 unterstützt wird.

Einschränkungen:

- Berichtsvariablen werden für eine vollständige Funktion definiert und beeinflussen die Verarbeitung aller REPORT-Blöcke und die Tabellen, die sie verarbeiten. 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 in einer Funktion verwendet werden können.

Bedingte Logik und Schleifen in einer Makrodatei

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

- „Bedingte Logik“
- „Schleifenkonstrukte“ auf Seite 79

Bedingte Logik

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.

Die Syntaxregeln für einen IF-Block werden durch die Position des Blocks in der Makrodatei 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, daß 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
```

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
```

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 block
...
  %IF block
```

Ausnahme: Geben Sie keinen ROW-Block in einem IF-Block an, wenn sich der IF-Block in einem REPORT-Block befindet.

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 Zeichenfolgenvergleiche wie in den Bedingungen angegeben durchzuführen. Wenn jedoch die beiden folgenden Bedingungen zutreffen, führt Net.Data einen numerischen Vergleich durch:

- Die Bedingung ist eine Binäroperation (<, >, <=, >=, !=, ==).
- Beide Ausdrücke in der Bedingung stellen ganze Zahlen dar. Das heißt, die Ausdrücke sind Zeichenfolgen aus Ziffern, denen wahlfrei ein Zeichen '+' oder '-' vorangestellt sein kann. Die Zeichenfolge darf keine Nichtziffernzeichen außer '+' bzw. '-' enthalten.

Beispiele für gültige Zeichenfolgen:

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

Beispiele für ungültige Zeichenfolgen:

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

Net.Data wertet den IF-Block 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: Eine Makrodatei, die IF-Blöcke in anderen Blöcken enthält

```
%{ 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)
    %}
%}
```

Einschränkung: Net.Data unterstützt keinen numerischen Vergleich von nichtganzzahligen Zahlen.

Schleifenkonstrukte

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 HTML-IF-Blöcken angeben, und Sie können sie verschachteln. Die Syntax eines WHILE-Blocks wird im Kapitel über die Sprachkonstrukte im Handbuch *Net.Data Reference* dargestellt.

Net.Data verarbeitet den WHILE-Block in exakt der gleichen Weise wie auch der IF-Block verarbeitet wird, jedoch wird die Bedingungsliste bei jedem Durchlaufen der Schleife erneut ausgewertet. Und wie bei jedem bedingten Schleifenkonstrukt kann die Verarbeitung zu einer Endlosschleife führen, wenn die Bedingung nicht korrekt codiert wurde.

Beispiel: Eine Makrodatei 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. Auch mit der SYSTEM-Sprachumgebung können Sie ein Programm ausführen oder einen Befehl absetzen.

Bei 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. Sie müssen der Net.Data-Initialisierungsdatei eine Anweisung ENVIRONMENT hinzufügen, um einer benutzerdefinierten Sprachumgebung ein Serviceprogramm zuzuordnen. Beim ersten Funktionsaufruf für einen FUNCTION-Block, der den Sprachumgebungs-namen angibt, lädt Net.Data ein Serviceprogramm und führt es einmal aus. Bei nachfolgenden Funktionsaufrufen für FUNCTION-Blöcke, die den gleichen Sprachumgebungs-namen angeben, führt Net.Data das geladene Serviceprogramm lediglich aus.

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

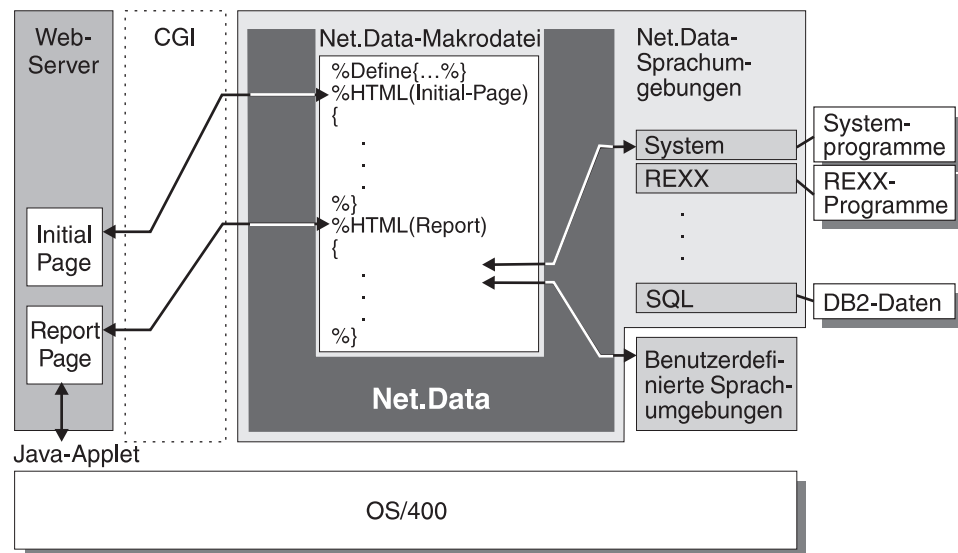


Abbildung 8. Die Net.Data-Sprachumgebungen

Ausführliche Informationen zum Erstellen einer benutzerdefinierten Sprachumgebung finden Sie im Handbuch *Net.Data Language Environment Reference*.

In den folgenden Abschnitten werden die Sprachumgebungen beschrieben, die in Net.Data enthalten sind. Dies schließt die Konfiguration und Verwendung der Sprachumgebungen ein.

- „REXX-Sprachumgebung“
- „SQL-Sprachumgebung“ auf Seite 85
- „SYSTEM-Sprachumgebung“ auf Seite 98

REXX-Sprachumgebung

Die REXX-Sprachumgebung ermöglicht die Interpretation von REXX-Programmen, die in einem FUNCTION-Block des Net.Data-Makros angegeben sind, sowie die Ausführung von externen REXX-Programmen, die in einer separaten Datei gespeichert sind.

Konfigurieren der REXX-Sprachumgebung

Führen Sie die folgenden Schritte aus, um die REXX-Sprachumgebung zu konfigurieren. Wenn Sie keine Net.Data-Initialisierungsdatei erstellen, wird die REXX-Sprachumgebung standardmäßig aktiviert. Es ist keine weitere Konfiguration erforderlich.

1. Wenn Sie eine Initialisierungsdatei erstellt haben und die REXX-Sprachumgebung verwenden möchten, müssen Sie die folgende Konfigurationsanweisung in die Initialisierungsdatei einfügen:

```
ENVIRONMENT(DTW_REXX)/QSYS.LIB/QTCP.LIB/QTMHREXX.SRVPGM ( )
```

Informationen zur Net.Data-Initialisierungsdatei, einschließlich Anweisungen zur Konfiguration der Umgebung, finden Sie im Abschnitt „Konfigurieren von Net.Data“ auf Seite 5.

2. Stellen Sie sicher, daß sich externe REXX-Programme im Dateisystem QSYS.LIB befinden.

Aufrufen von externen REXX-Programmen

Verwenden Sie zum Aufrufen von externen REXX-Programmen einen FUNCTION-Block mit dem folgenden Format:

```
%EXEC{ REXX-file-name [optional parameters] %}
```

Beispiel:

```
%FUNCTION(DTW_REXX) rexx1() {  
%EXEC{ /QSYS.LIB/REXX.LIB/REXXSRC.FILE/TREXX.MBR %}  
%}
```

Übergeben von Parametern

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`.

Verwenden der REXX-Anweisung SAY mit OS/400 V3R2 oder V3R7

Wenn Sie OS/400 V3R2 oder V3R7 ausführen und ein REXX-Programm die REXX-Anweisung SAY verwendet, um auf 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.

Sicherheit

Stellen Sie sicher, daß die Benutzer-ID, unter der Net.Data ausgeführt wird, Lese- und Schreibzugriffsrechte für die Datei mit dem externen REXX-Programm sowie für alle Objekte besitzt, die das REXX-Programm verwendet.

Optimieren der Leistung

Beachten Sie die folgenden Hinweise, um die Leistung Ihrer Net.Data-Anwendung zu optimieren:

- Beschränken Sie die Anzahl der REXX-Programme, die aufgerufen werden, indem Sie REXX-Programme kombinieren. 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 immer in einer neuen Aktivierungsgruppe ausgeführt wird. Der Interpreter muß jedes Mal aktiviert werden, wenn Net.Data den Interpreter aufruft, um ein REXX-Programm auszuführen.
- Speichern Sie das REXX-Programm in einer externen Datei, statt es inline in das Net.Data-Makro einzufügen.
- Übergeben Sie Eingabeparameter direkt an ein REXX-Programm, indem Sie globale Net.Data-Variablen definieren und auf die Variablen verweisen. Rufen Sie bei Inline-REXX-Programmen die globalen Variablen direkt in Ihrer REXX-Quelle auf.

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
end
%}

%HTML(REPORT){
  @genTable(myTable)
  @DTW_TB_TABLE(myTable)
%}
```

Ergebnisse:

| COL1 | COL2 |
|---------|---------|
| [1 1] | [1 2] |
| [2 1] | [2 2] |
| [3 1] | [3 2] |

Wenn das Web-Makro in Bibliothek NETDATA, Datei REXXMAC und Member REXX1 gespeichert ist, wird auf das Makro verwiesen, indem die folgende URL-Adresse in einem Browser geladen wird:

```
http://hostname/cgi-bin/db2www/qsys.lib/netdata.lib/rexxmac.file/
  rexx1.mbr/report
```

SQL-Sprachumgebung

Verwenden Sie die SQL-Sprachumgebung, um mit Hilfe von DB2 SQL-Anweisungen auszuführen. Jede gültige SQL-Anweisung kann an die SQL-Sprachumgebung übergeben werden, sofern es sich bei den SQL-Anweisungen um gültige DB2 für OS/400-Befehle handelt.

Konfigurieren der SQL-Sprachumgebung

Führen Sie die folgenden Schritte aus, um die SQL-Sprachumgebung zu konfigurieren.

1. Erstellen Sie einen Verzeichniseintrag für die lokale Datenbank im relationalen Datenbankverzeichnis (d. h. einen Verzeichniseintrag mit einer fernen Position *LOCAL). Dies zusätzlich zu allen fernen Datenbanken, auf die die SQL-Sprachumgebung zugreifen wird. Fügen Sie den Eintrag mit dem Befehl Add Relational Database Directory Entry (ADDRDBDIRE) hinzu.

Wenn Sie keine Net.Data-Initialisierungsdatei erstellen, wird die SQL-Sprachumgebung standardmäßig aktiviert. Es ist keine weitere Konfiguration erforderlich.

2. Ändern Sie die Net.Data-Initialisierungsdatei.
 - a. Wenn Sie eine Initialisierungsdatei erstellt haben und die SQL-Sprachumgebung verwenden möchten, müssen Sie die folgende Konfigurationsanweisung in die Initialisierungsdatei einfügen. Der Text dieser Umgebungsanweisung muß in der Initialisierungsdatei vollständig in einer Zeile stehen. Er wird hier zur besseren Lesbarkeit in mehreren Zeilen dargestellt.

```
ENVIRONMENT(DTW_SQL) /QSYS.LIB/QTCP.LIB/QTMSQL.SRVPGM ( )  
  (IN DATABASE, LOGIN, PASSWORD, TRANSACTION_SCOPE, SHOWSQL,  
   DTW_SET_TOTAL_ROWS, DB_CASE, START_ROW_NUM, RPT_MAX_ROWS,  
   OUT DTWTABLE, SQL_CODE, TOTAL_ROWS)
```

Sie müssen nicht alle oben angegebenen Variablen in der Umgebungsanweisung angeben. Wenn Sie keine Variable wie DTW_SET_TOTAL_ROWS und TOTAL_ROWS verwenden, können Sie sie aus der Umgebungsanweisung entfernen. Eine andere Möglichkeit zur Übergabe dieser Variablen an die SQL-Sprachumgebung besteht darin, die Variablen im Net.Data-Funktionsaufruf in Ihrem Makro zu übergeben. Informationen zu den Variablen der Sprachumgebung finden Sie im Handbuch *Net.Data Reference*.

- b. 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, wie in Tabelle 1 auf Seite 86 gezeigt.

Tabelle 1. Konfigurationsvariablen der SQL-Sprachumgebung

| Konfigurationsvariable | Beschreibung |
|------------------------|---|
| DTW_SQL_ISOLATION | <p>Legt fest, in welchem Umfang Datenbankoperationen, die von der SQL-Sprachumgebung ausgeführt werden, von gleichzeitig ablaufenden Prozessen isoliert werden. Gültige Werte sind:</p> <p>DTW_SQL_NO_COMMIT</p> <p>DTW_SQL_READ_UNCOMMITTED</p> <p>DTW_SQL_READ_COMMITTED</p> <p>DTW_SQL_REPEATABLE_READ</p> <p>DTW_SQL_SERIALIZABLE</p> <p>Der Standardwert ist DTW_SQL_READ_UNCOMMITTED.</p> |
| DTW_SQL_NAMING_MODE | <p>Legt fest, wie ein Tabellenname in einer SQL-Anweisung angegeben werden kann. Gültige Werte sind:</p> <p>SQL_NAMING</p> <p>SYS_NAMING</p> <p>Der Standardwert ist SQL_NAMING.</p> |

Weitere Informationen zur Net.Data-Initialisierungsdatei einschließlich Anweisungen zu Umgebungsvariablen und Konfigurationsvariablenanweisungen finden Sie im Abschnitt „Konfigurieren von Net.Data“ auf Seite 5.

Ausführen der SQL-Sprachumgebung unter COMMIT-Steuerung

Die SQL-Sprachumgebung wird standardmäßig unter COMMIT-Steuerung ausgeführt und befolgt alle Regeln zur Verwaltung der COMMIT-Steuerung.

- Zeichnen Sie alle Dateien oder Tabellen, auf die über DTW_SQL zugegriffen wird, in einem Journal auf, es sei denn, die SQL-Anweisung lautet SELECT.

Wenn Sie auf eine SQL-Tabelle in einer Datensammlung zugreifen, wird sie über die native SQL-Unterstützung auf dem System IBM AS/400 automatisch aufgezeichnet, so daß keine explizite Maßnahme erforderlich ist.

- Ändern Sie wahlfrei die COMMIT-Stufe, indem Sie DTW_SQL_ISOLATION in der Net.Data-Initialisierungsdatei angeben. Nähere Informationen zu den Isolationsstufen, die die SQL-Sprachumgebung unterstützt, finden Sie im Abschnitt „Konfigurieren der SQL-Sprachumgebung“ auf Seite 85.

Verwalten von Datenbankverbindungen für ferne Datenbanken

Sie können Verbindungen zu bis zu 50 lokalen oder fernen Datenbanken gleichzeitig herstellen. Die SQL-Sprachumgebung hält die Verbindungen für die Dauer des HTTP-Server-Prozeßjobs, unter dem Net.Data ausgeführt wird, aktiv. Dadurch, daß die Verbindungen aktiv gehalten werden, wird der Datenbankzugriff beschleunigt, nachdem die einleitende Verbindung zur Datenbank hergestellt wurde. Sie können Fehler vermeiden und die Leistung verbessern, indem Sie die folgenden Punkte berücksichtigen:

- Stellen Sie sicher, daß Sie Ihre Datenbankverbindungen auf der Grundlage der Einstellung für den Gültigkeitsbereich einer Transaktion planen:

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, 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. Planen Sie Ihre Verbindungen auf der Grundlage der folgenden Regeln:

- 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 definiert ist. Dies ist die Standardeinstellung. Die SQL-Sprachumgebung führt automatisch eine ROLLBACK-Operation aus, und der SQL_CODE -752 wird zurückgegeben, was darauf hinweist, daß die Verbindung nicht geändert werden konnte.
- Verbessern Sie die Leistung, indem Sie die Anzahl der Benutzer-IDs verringern, die eine Verbindung zur gleichen fernen Datenbank herstellen.

Wenn die SQL-Sprachumgebung eine Verbindung zu einem fernen System herstellt, wird der Verbindung eine Benutzer-ID zugeordnet. Wenn bei einer nachfolgenden Net.Data-Abfrage die Benutzer-ID nicht mit der der Verbindung zugeordneten Benutzer-ID übereinstimmt, wird die Verbindung beendet und eine neue Verbindung zur Datenbank hergestellt (dies ist nur der Fall, wenn der Gültigkeitsbereich einer Transaktion SINGLE ist).

Codieren Sie zur Erhöhung der Leistung die Benutzer-ID fest, oder verwenden Sie die gleiche Benutzer-ID, wenn Sie SQL-Anweisungen an eine ferne Datenbank absetzen. Beim Zugriff auf die lokale Datenbank werden Benutzer-ID und Kennwort ignoriert.

Gespeicherte Prozeduren

Eine gespeicherte Prozedur ist ein kompiliertes Programm, das auf dem lokalen oder fernen DB2-Server gespeichert ist und SQL-Anweisungen ausführen kann. In Net.Data werden gespeicherte Prozeduren von Net.Data-Funktionen mit Hilfe der SQL-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.

In diesem Abschnitt werden folgende Themen behandelt:

- „Syntax für gespeicherte Prozeduren“
- „Aufrufen einer gespeicherten Prozedur“ auf Seite 89
- „Übergeben von Parametern“ auf Seite 90
- „Verarbeiten von Ergebnismengen“ auf Seite 90

Syntax für gespeicherte Prozeduren

Die Syntax für die gespeicherte Prozedur verwendet die Anweisung FUNCTION, die Anweisung CALL und wahlfrei einen REPORT-Block.

```
%function (dtw_sql) function_name ([IN datatype arg1, INOUT datatype arg2,  
    OUT tablename, ...])  
    CALL stored_procedure  
[%REPORT(resultsetname...)]
```

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 2 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 ein Name, der eine von der gespeicherten Prozedur zurückgegebene Ergebnismenge einem REPORT-Block zuordnet. Wenn dieser Parameter angegeben ist, muß er mit dem zugehörigen Parameternamen für tablename übereinstimmen.

Tabelle 2. Datentypen für gespeicherte Prozeduren

| | | |
|-----------------|----------|------------|
| CHAR | FLOAT | TIME |
| DATE | GRAPHIC | TIMESTAMP |
| DECIMAL | INTEGER | VARCHAR |
| DOUBLE | REAL | VARGRAPHIC |
| DOUBLEPRECISION | SMALLINT | |

Aufrufen einer gespeicherten Prozedur

Gehen Sie wie folgt vor, um eine gespeicherte Prozedur aufzurufen:

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.

```
%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. Ein Parameter wird an eine gespeicherte Prozedur übergeben, indem der Parametername mit dem Schlüsselwort IN angegeben wird. Wenn eine gespeicherte Prozedur den Parameter aktualisiert, müssen Sie den Parameter für den zurückzugebenden Wert mit dem Schlüsselwort INOUT bzw. OUT übergeben. Der Datentyp, der für einen Parameter angegeben wird, muß mit dem übereinstimmen, den die gespeicherte Prozedur erwartet.

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 angeben, ob eine oder mehrere Ergebnismengen von einer gespeicherten Prozedur zurückgegeben werden. Die Ergebnismengen können in Net.Data-Tabellen zur weiteren Verarbeitung innerhalb Ihres Makros gespeichert oder mit Hilfe eines REPORT-Blocks angezeigt werden. Sie müssen jeder durch eine gespeicherte Prozedur generierten Ergebnismenge einen Namen zuordnen. Dies geschieht durch die Angabe von Parametern in der FUNCTION-Anweisung. 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 im Standardformat für Net.Data-Berichte anzeigen, indem Sie keinen REPORT-Block für die Ergebnismenge definieren.
- Sie können eine Ergebnismenge einem REPORT-Block zuweisen, damit Net.Data die Ergebnismenge in einem Bericht anzeigt. In diesem Fall können Sie 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 die Daten später in der Makrodatei verwenden 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 hinsichtlich der Verwendung mehrerer REPORT-Blöcke finden Sie im Abschnitt „Richtlinien und Einschränkungen für mehrere REPORT-Blöcke“ auf Seite 75.

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  
%}
```

Wenn nur eine Ergebnismenge zurückgegeben und ein REPORT-Block für die Verarbeitung der Anzeige angegeben werden soll:

Verwenden Sie die folgende Syntax:

```
%function (dtw_sql) function_name (OUT tablename) {  
    CALL stored_procedure  
    %REPORT (resultsetname) {  
        ...  
    %}  
%}
```

Beispiel:

```
%function (dtw_sql) mystoredproc (OUT mytable1) {  
    CALL myproc  
    %REPORT (mytable1) {  
        ...  
        %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 { ... %}  
        ...  
    %}  
%}
```

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 = "MULTIPLE"  
%function (dtw_sql) mystoredproc (OUT mytable1, mytable2) {  
    CALL myproc  
%}
```

Wenn mehrere Ergebnismengen zurückgegeben und REPORT-Blöcke für die Verarbeitung der Anzeige angegeben werden sollen:

Jeder Ergebnismenge wird ein eigener REPORT-Block zugeordnet. Verwenden Sie die folgende Syntax:

```
%function (dtw_sql) function_name (OUT tablename1, tablename2) {  
    CALL stored_procedure  
    %REPORT (resultsetname1)  
        ...  
    %row { ... %}  
        ...  
    %}  
    %REPORT (resultsetname2)  
        ...  
    %row { ... %}  
        ...  
    %}  
%}
```

Beispiel:

```
%function (dtw_sql) mystoredproc (OUT mytable1, mytable2) {  
  CALL myproc  
  
  %REPORT (mytable1) {  
    ...  
    %row { ... %}  
    ...  
    %}  
  
  %REPORT(mytable2) {  
    ...  
    %row { ... %}  
    ...  
    %}  
  %}
```

Einschränkungen der SQL-Sprachumgebung

Berücksichtigen Sie die folgenden Einschränkungen, wenn Sie Ihre Umgebung planen:

- Die SQL-Sprachumgebung darf in folgenden Fällen nicht verwendet werden:
 - Es wird eine benutzerdefinierte Sprachumgebung erstellt, die die Klassenbibliothek für den Datenbankzugriff oder SQL Call Level Interface verwendet.
 - Auf die benutzerdefinierte Sprachumgebung wird außerdem in einem Makro verwiesen.
- Sie dürfen in einer EXEC-Anweisung keine SQL-Anweisungen an die SQL-Sprachumgebung übergeben.

Sicherheit

Stellen Sie sicher, daß die Benutzer-ID, unter der Net.Data ausgeführt wird, Zugriffsrechte für die Datenbank auf dem Rechner besitzt, auf dem sich der HTTP-Server befindet. Diese Rechte sind auch für Datenbankdateien und Journale (bei aktiver COMMIT-Steuerung) erforderlich.

Beim Zugriff auf ferne Datenbanken werden die Net.Data-Variablen LOGIN (Benutzer-ID) und PASSWORD verwendet, um zu bestimmen, auf welche Datenbankressourcen zugegriffen werden kann. Diese Variablen werden ignoriert, wenn auf die Datenbank auf dem Rechner zugegriffen wird, auf dem sich der HTTP-Server befindet, weil der Zugriff auf Objekte vom Benutzerprofil bestimmt wird, unter dem Net.Data ausgeführt wird.

Optimieren der Leistung

Informationen zu Überlegungen hinsichtlich der DB2-Leistung finden Sie im Handbuch *DB2 for AS/400 SQL Programming Guide*. 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. In den folgenden Abschnitten werden wichtige Aspekte der Verfahren für Datenbanken und die SQL-Sprachumgebung hervorgehoben:

Datenbankverfahren

Die folgende Zusammenfassung nennt einige der einfachsten Datenbankverfahren, mit denen der Zugriff auf die Datenbank verbessert werden kann:

- Vermeiden Sie numerische Umwandlungen. Wenn ein Spaltenwert und ein Literalwert verglichen werden, versuchen Sie, die gleichen Datentypen und Attribute anzugeben. DB2 für AS/400 verwendet keinen Index für die benannte Spalte, wenn der Literalwert eine höhere Genauigkeit hat als die Spalte. Wenn die beiden zu vergleichenden Elemente unterschiedliche Datentypen aufweisen, muß DB2 für AS/400 einen der beiden Werte umwandeln, was zu Ungenauigkeiten führen kann (aufgrund der beschränkten Rechengenauigkeit).

EDUCLVL ist beispielsweise ein Halbwortganzzahlenwert (SMALLINT). Geben Sie folgendes an:

```
... WHERE EDUCLVL < 11 AND EDUCLVL >= 2
```

Und nicht:

```
... WHERE EDUCLVL < 1.1E1 AND EDUCLVL > 1.3
```

- Vermeiden Sie das Auffüllen von Zeichenfolgen. Versuchen Sie, die gleiche Datenlänge zu verwenden, wenn Sie einen Zeichenfolgenspaltenwert mit fester Länge mit einem Literalwert vergleichen. DB2 für AS/400 verwendet keinen Index, wenn der Literalwert länger ist als die Spaltenlänge.

EMPNO beispielsweise ist CHAR(6) und DEPTNO CHAR(3). Geben Sie folgendes an:

```
... WHERE EMPNO > '000300' AND DEPTNO < 'E20'
```

Und nicht:

```
... WHERE EMPNO > '000300 ' AND DEPTNO < 'E20 '
```

- Vermeiden Sie die Verwendung von LIKE-Mustern, die mit % oder _ anfangen. Das Prozentzeichen (%) und das Unterstreichungszeichen (_) geben bei Verwendung im Muster eines LIKE-Vergleichselements eine Zeichenfolge an, die ähnlich ist wie der Spaltenwert von Zeilen, die Sie auswählen möchten. Bei Verwendung zur Angabe von Zeichen in der Mitte oder am Ende einer Zeichenfolge können LIKE-Muster Indizes nutzen. Beispiel:

```
... WHERE LASTNAME LIKE 'J%SON%'
```

Bei Verwendung am Anfang einer Zeichenfolge können LIKE-Muster DB2 für AS/400 jedoch daran hindern, Indizes zu verwenden, die in der Spalte LASTNAME definiert sein könnten, um die Anzahl der gelesenen Zeilen zu beschränken. Beispiel:

```
... WHERE LASTNAME LIKE '%SON'
```

Vermeiden Sie die Verwendung dieser Symbole am Anfang von Zeichenfolgen, vor allem, wenn Sie auf eine besonders große Tabelle zugreifen.

Verfahren für die SQL-Sprachumgebung

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 im Abschnitt „Transaktionsverwaltung mit permanenten Makros“ auf Seite 101.
- 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.

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 Zeichenfolgenwerte 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>
%}

```

Wenn das Web-Makro in Bibliothek NETDATA, Datei SQLMAC und Member SQL1 gespeichert ist, wird auf das Makro verwiesen, indem die folgende URL-Adresse in einem Browser geladen wird:

```

http://hostname/cgi-bin/db2www/qsys.lib/netdata.lib/sqlmac.file/
sql1.mbr/report

```

SYSTEM-Sprachumgebung

Die SYSTEM-Sprachumgebung unterstützt Aufrufe von externen Programmen in RPG, COBOL und C sowie die Ausführung von CL-Befehlen, die in einer EXEC-Anweisung im FUNCTION-Block angegeben sind. Die SYSTEM-Sprachumgebung interpretiert die EXEC-Anweisung, indem der angegebene Programmname oder Befehl mit Parametern mit dem C-Funktionsaufruf `system()` zur Ausführung an das Betriebssystem übergeben wird.

Konfigurieren der SYSTEM-Sprachumgebung

Wenn Sie keine Net.Data-Initialisierungsdatei erstellen, wird die SYSTEM-Sprachumgebung standardmäßig aktiviert. Es ist keine weitere Konfiguration erforderlich.

Wenn Sie eine Initialisierungsdatei erstellt haben und die SYSTEM-Sprachumgebung verwenden möchten, müssen Sie die folgende Konfigurationsanweisung in die Initialisierungsdatei einfügen.

```
ENVIRONMENT(DTW_SYSTEM) /QSYS.LIB/QTCP.LIB/QTMSYS.SRVPGM ( )
```

Informationen zur Net.Data-Initialisierungsdatei, einschließlich Anweisungen zur Konfiguration der Umgebung, finden Sie im Abschnitt „Konfigurieren von Net.Data“ auf Seite 5.

Übergeben von Parametern

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')
  %}
%}
```

Der Verweis auf die Net.Data-Variable INPARAM1 wird aufgehoben, und die Variable wird an das Programm übergeben. Die Parameter werden dem Programm auf gleiche Weise übergeben wie bei einem Aufruf des Programms über die Anzeige zur Befehlseingabe (Command Entry). 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 3 beschrieben werden, aktualisiert:

Tabelle 3. 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.

Sicherheit

Stellen Sie sicher, daß die Benutzer-ID, unter der Net.Data ausgeführt wird, Zugriffsrechte für die Programmausführung sowie für alle Objekte besitzt, auf die das Programm zugreift.

Optimieren der Leistung

Übergeben Sie Eingabeparameter direkt an das Programm, das die SYSTEM-Sprachumgebung aufruft, indem Sie globale Net.Data-Variablen definieren und darauf verweisen.

Beispiel für die SYSTEM-Sprachumgebung

Das folgende Beispiel zeigt ein Makro, das eine Funktionsdefinition mit den drei Parametern P1, P2 und P3 enthält. P1 ist ein Eingabeparameter (IN), P2 und P3 sind Ausgabeparameter (OUT). Die Funktion ruft ein Programm, UPDPGM, auf, das den Parameter P2 mit dem Wert von P1 aktualisiert und P3 auf eine Zeichenfolge setzt. Vor der Verarbeitung der Anweisung im %EXEC-Block speichert die Sprachumgebung DTW_SYSTEM P1 und den zugehörigen Wert im Umgebungsbereich.

```
%DEFINE {
    MYPARM2      = "ValueOfParm2"
    MYPARM3      = "ValueOfParm3"
}%
%FUNCTION(DTW_SYSTEM) sys1 (IN P1, OUT P2, P3) {
    %EXEC {
        /QSYS.LIB/NETDATA.LIB/UPDPGM.PGM
    }
}%

%HTML(upd1) {
    <P>
    Passing data to a program. The current value
    of MYPARM2 is "$(MYPARM2)", and the current value of MYPARM3 is
    "$(MYPARM3)". Now we invoke the Web macro function.

    @sys1("ValueOfParm1", MYPARM2, MYPARM3)

    <P>
    After the function call, the value of MYPARM2 is "$(MYPARM2)",
    and the value of MYPARM3 is "$(MYPARM3)".
}%
```

Wenn das Web-Makro in Bibliothek NETDATA, Datei SYSMAC und Member SYS1 gespeichert ist, wird auf das Makro verwiesen, indem die folgende URL-Adresse in einem Browser geladen wird:

```
http://hostname/cgi-bin/db2www/qsys.lib/netdata.lib/sysmac.file/
sys1.mbr/upd1
```

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 nichtpermanenten 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 nichtpermanenten 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 im Abschnitt „Aufrufen eines permanenten Makros“ auf Seite 31.

In diesem Kapitel werden die folgenden Themen beschrieben:

- „Informationen zu permanenten Makros“
- „Definieren einer Transaktion“ auf Seite 102
- „Beispiel für ein permanentes Makro“ auf Seite 112

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-HTTPSession“, 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 103
- „Angabe der Makro-HTML-Blöcke in einer Transaktion“ auf Seite 104
- „Beenden einer Transaktion“ auf Seite 109
- „Definieren des Gültigkeitsbereichs einer Variablen in einer Transaktion“ auf Seite 110
- „Angabe von COMMIT- und ROLLBACK-Operationen in einer Transaktion“ auf Seite 111

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, daß 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: Jede der folgenden URL-Adressen muß als fortlaufende Zeichenfolge in Ihrem Code angegeben werden; sie werden hier nur zur besseren Lesbarkeit 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 Net.Data-Programmdatei 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 im Abschnitt „MACRO_PATH“ auf Seite 13.

block Gibt den Namen des HTML-Blocks in der angegebenen Net.Data-Makrodatei an.

method

Gibt die für das Formular verwendete HTML-Methode an. Hierfür wird METHOD=POST empfohlen.

?name=val&...

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="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>
</form>
%}
```

Beenden einer Transaktion

Sie beenden einer 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 nichtpermanente 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 nichtpermanenten 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/pcgi1.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.

Anhang A. Problemanalyse

In diesem Abschnitt werden Verfahren zur Problemanalyse beschrieben, die Sie zum Beheben von Fehlern bei Net.Data einsetzen können.

Bei den folgenden Problemen wird vorausgesetzt, daß das CGI-BIN-Programmobjekt von Net.Data, DB2WWW, in eine Bibliothek namens WWWCGI verschoben wurde, in der sich CGI-BIN-Programme befinden.

- **Symptom:** Error 500:

```
Bad script request -- '/QSYS.LIB/WWWCGI.LIB/DB2WWW.PGM/  
QSYS.LIB' not executable
```

Ursache: Eine Exec-Regel ist falsch.

```
Exec /QSYS.LIB/WWWCGI.LIB/DB2WWW.PGM/*  
Exec /qsys.lib/wwwcgi.lib/db2www.pgm/*
```

Lösung: Geben Sie eine Exec-Regel an, die nur den Pfad zum Programm DB2WWW angibt. Beispiel:

```
Exec /QSYS.LIB/WWWCGI.LIB/*  
Exec /qsys.lib/wwwcgi.lib/*
```

- **Symptom:** Error 404:

```
Not found - file doesn't exist or is read protected  
Even tried multi"
```

Ursache: Eine Exec-Regel fehlt.

Lösung: Geben Sie eine Exec-Regel an, die den Pfad zum Programm DB2WWW sowohl in Groß- als auch in Kleinbuchstaben angibt. Beispiel:

```
Exec /QSYS.LIB/WWWCGI.LIB/*  
Exec /qsys.lib/wwwcgi.lib/*
```

- **Symptom:** Error 403:

```
Forbidden - by rule
```

Ursache: Eine Map- oder Exec-Regel fehlt oder ist falsch.

Lösung: Geben Sie eine Map- und Exec-Regel für das Programm DB2WWW sowohl in Groß- als auch in Kleinbuchstaben an. Beispiel:

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

- **Symptom:** Die Konfigurationsanweisungen in der Web-Server-Konfigurationsdatei sind richtig, Net.Data verarbeitet jedoch Makrodateien nicht richtig. Es gibt mehrere mögliche Ursachen und Lösungen:

- **Ursache:** Die Map-, Exec- oder Pass-Regeln haben nicht die richtige Reihenfolge.

Lösung: Wenn eine URL-Adresse für eine Map-, Exec- oder Pass-Regel ausgewertet wird, wird sie auf Basis der ersten übereinstimmenden Regel verarbeitet.

Stellen Sie sicher, daß die auszuwertende Anweisung vor Erreichen der gewünschten Regel nicht erneut zugeordnet oder geändert wird. Stellen Sie außerdem sicher, daß die Konfigurationsdatei des Benutzers keine Anweisung `Pass /*` enthält.

- **Ursache:** Das Benutzerprofil QTMHHTTP1 hat nicht die erforderliche Berechtigung für den Zugriff auf Net.Data-Makros.

Lösung: Alle CGI-BIN-Programme werden unter dem Benutzerprofil QTMHHTTP1 ausgeführt. Dem Benutzerprofil QTMHHTTP1 muß die Berechtigung erteilt werden, auf alle Objekte zugreifen zu können, auf die Net.Data bei der Verarbeitung eines Net.Data-Makros zugreift.

- **Ursache:** Die Pfadanweisungen in der Net.Data-Initialisierungsdatei stimmen nicht.

Lösung: Stellen Sie sicher, daß entweder die Objektverweise vollständig qualifiziert sind oder daß die Net.Data-Initialisierungsdatei die richtigen Pfadanweisungen enthält. Net.Data verwendet die Pfadanweisungen in der Initialisierungsdatei (falls vorhanden), um im verarbeiteten Net.Data-Makro Verweise auf Net.Data-Makros oder Programmdateien aufzulösen. Wenn Objektverweise nicht vollständig qualifiziert sind und die Pfadanweisungen in der Initialisierungsdatei nicht stimmen, gibt Net.Data an, daß das Objekt, auf das verwiesen wird, nicht gefunden wurde.

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.

```
%{***** Sample Macro *****}
*   FileName = sqlsamp1.d2w                                     *
*   Description:                                               *
*       This Net.Data macro file 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 "sqlsamp1.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 sqlsamp1.hti.     *
*****%}
%FUNCTION(DTW_SQL) queryDB() {
    SELECT * 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>$(V2)
%}
</select>
%}
%}
```

```
%{*****
* Function: fname          Language Environment: SQL          *
* Description: Queries the table designated by the variable myTable *
*               for additional information about the employee identified *
*               by variable emp_name.                             *
*****%}
%FUNCTION(DTW_SQL) fname(){
SELECT EMPNME, PHONENO, JOB FROM $(myTable) WHERE EMPNME='$(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
}%
%}
```

```
%{*****
* 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 %}
=====
%{***** 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 = "MRZ.EMPLOYEE"
%}

%{      End of include file %}

```

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.

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.

Marken

Folgende Namen sind in gewissen Ländern Marken der IBM Corporation:

| | |
|---|----------------------|
| AIX | IMS |
| AS/400 | Language Environment |
| CBIDO | MVS/ESA |
| CBPDO | Net.Data |
| CICS | OpenEdition |
| CustomPac | Operating System/400 |
| DB2 | OS/2 |
| DB2 Universal Database | OS/390 |
| DataJoiner | OS/400 |
| Distributed Relational Database Architecture | RACF |
| DRDA | SystemPac |
| IBM | |

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 höherer 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 Verbesserung der Leistung über CGI-Prozesse: ICAPI, GWAPI, ISAPI und NSAPI.

API. Anwendungsprogrammierschnittstelle. Net.Data unterstützt drei eigene 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, verwendet werden und werden zusammen mit der HTML-Seite geladen.

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. Ein Standardverfahren zur Übergabe der Steuerung durch einen Web-Server an ein Anwendungsprogramm und zum anschließenden Datenempfang

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

DBMS. Datenbankverwaltungssystem (Database Management System)

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

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

ICS. Internet Connection Server

ICSS. Internet Connection Secure Server

Internet. Ein internationales, öffentliches TCP/IP-Computernetzwerk

Internet Connection Secure Server. Gesicherter Web-Server von IBM

Internet Connection Server. Nicht gesicherter Web-Server von IBM

Intranet. Ein TCP/IP-Netzwerk innerhalb eines 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 interpretierte Programmiersprache

Permanenz (persistence). Der Zustand, daß ein zugeordneter Wert eine ganze Transaktion lang beibehalten wird, die mehrere Net.Data-Aufrufe beinhaltet. 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. Einige Sprachumgebungen werden mit Net.Data ausgeliefert, wie zum Beispiel REXX, Perl und Oracle. Sie können ferner Ihre eigenen Sprachumgebungen erstellen.

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 einer Makrodatei
 - Darstellung 33
 - Deklaration 33
- Arten von Variablen 44
- Aufrufen gespeicherter Prozeduren 88, 89
- Aufrufen von Funktionen 60
- Aufrufen von Net.Data
 - CGI 27
 - Direktanforderung 27
 - Formulare 27, 31, 106
 - HTML-Blöcke 67
 - Makroanforderung 27
 - mit einer Makrodatei 27
 - Programmverbindungen (Links) 27, 31, 106
 - Syntax 27
 - Übersicht 27
 - URL-Adressen 27, 31, 106
- Ausführbare Variablen 46
- Ausgabe, Inaktivieren für Standardberichte 70
- Authentifizierung, Sicherheit 24

B

- Bedingt
 - Logik, IF-Blöcke 76
 - Variablen 45
- Beispielmakro 116
- Bemerkungen 120
- Benutzerdefinierte Funktionen 54
- Berechtigung
 - Net.Data-Dateien, Angeben von Zugriffsrechten 19
 - Sicherheit 25
- Berichtsformate anpassen 71
- Berichtsvariablen 52
- BLOBs
- Blöcke, Makrodatei 36

C

- CGI-BIN-Bibliothek, Net.Data-Programmdateiobjekt kopieren 6

D

- Dateien, Angeben von Zugriffsrechten für Net.Data 19
- Datentypen, gültig für gespeicherte Prozeduren 88
- DEFINE-Block
 - Beschreibung 36
 - Definieren von Variablen 42
- Definieren von Variablen
 - DEFINE-Anweisung bzw. -Block 42

- Definieren von Variablen (*Forts.*)
 - HTML-Formularbefehle SELECT und INPUT 42
 - URL-Daten 43
- Deklarationsabschnitt, Makrodateistruktur 33
- DTW_SQL_ISOLATION 11
- DTW_SQL_NAMING_MODE 12
- DTWR_CLOSE_REGISTRIES 12

E

- ENVIRONMENT, Anweisungen
 - Beispiel 18
 - Beschreibung 17
 - Konfigurieren in der Initialisierungsdatei 17
 - Parameterliste 17
 - Serviceprogramm 17
 - Sprachumgebungsart 17
 - Syntax 17
- Ergebnismengen
 - mehrere 92
 - mehrere, Richtlinien und Einschränkungen 75
 - nur eine 91
 - verarbeiten, gespeicherte Prozeduren 90
- Erstellen einer Initialisierungsdatei 9

F

- Firewalls 21
- Formatieren der Datenausgabe 69
- Formulare
 - Aufrufen von Net.Data 27, 31, 106
 - in Web-Seiten zum Aufrufen von Net.Data 30
- FUNCTION-Block
 - Aufrufen von Funktionen 60
 - Beschreibung 37
 - Formatieren der Ausgabe 69
 - Geltungsbereich für Kennungen 41
 - Verarbeiten von Funktionsaufrufen 60
 - Verarbeiten von Variablenverweisen 60
- Funktionen
 - aufrufen 60
 - Aufrufen gespeicherter Prozeduren 88
 - benutzerdefinierte 54
 - Beschreibung 54
 - definieren 54
 - FUNCTION-Blocksyntax 54
 - MACRO_FUNCTION-Blocksyntax 55
- Funktionsaufrufe
 - Syntax 60
 - Verarbeitungsreihenfolge 60
- Fußzeilen, REPORT-Block 70

G

- Geltungsbereich
 - REPORT-Block 41
 - Variablen 41
- Geltungsbereich, Kennung
 - FUNCTION-Block 41
 - global 41
 - Makrodatei 41
 - ROW-Block 42
- Gespeicherte Prozeduren
 - Aufrufen in Makrodatei 88
 - gültige Datentypen 88
 - mehrere Ergebnismengen 92
 - mit einer Ergebnismenge 91
 - REPORT-Blöcke 91, 92
 - Schritte 89
 - Standardberichte 91, 92
 - Übergeben von Parametern 90
 - Verarbeiten von Ergebnismengen 90
- Globaler Geltungsbereich für eine Kennung 41
- Glossar 123

H

- HTML
 - Befehle für Tabellen 70
 - Blöcke
 - Aufrufen von Net.Data 67
 - Beispiel 67
 - Beschreibung 38
 - Verarbeitung 68
 - FORM, Knopf zum Übergeben (Submit) 68
 - Formulare
 - Aufrufen von Net.Data 27, 31, 106
 - Informationen zu 30
 - SELECT- und INPUT-Befehle, Definieren von Variablen 42
 - Generieren in einer Makrodatei 67
 - nicht erkannte Daten 68
 - Programmverbindungen (Links)
 - Aufrufen von Net.Data 27, 31, 106
 - Informationen zu 29

I

- IF-Blöcke 76
- IN-Parameter 60
- Informationen zu diesem Handbuch vii
- Initialisierungsdatei
 - aktualisieren 9
 - Beschreibung 8
 - ENVIRONMENT, Anweisungen 17
 - erstellen 9
 - Format 9
 - Konfigurationsvariablenanweisungen 10
 - Pfadanweisungen 13

INOUT-Parameter 60

K

- Konfigurationsvariablenanweisungen
 - Beschreibung 10
 - DTW_SQL_ISOLATION 11
 - DTW_SQL_NAMING_MODE 12
 - DTWR_CLOSE_REGISTRIES 12
 - Konfigurieren in der Initialisierungsdatei 10
- Konfigurieren von Net.Data
 - Initialisierungsdatei
 - aktualisieren 9
 - Beschreibung 8
 - ENVIRONMENT, Anweisungen 17
 - erstellen 9
 - Konfigurationsvariablenanweisungen 10
 - Pfadanweisungen 13
 - Übersicht 5
 - Zugriffsrechte für Net.Data-Dateien 19
- Kopfzeilen, REPORT-Block 70
- Kopieren des Net.Data-Programmdateiobjekts
 - in die CGI-BIN-Bibliothek 6
 - in mehrere Bibliotheken 6

L

- Listenvariablen 49

M

- MACRO_FUNCTION-Block
 - Aufrufen von Funktionen 60
 - Syntax 55
 - Verarbeiten von Funktionsaufrufen 60
 - Verarbeiten von Variablenverweisen 60
- Makroanforderung
 - Beispiele 27
 - Beschreibung 27
 - Syntax 27
- Makrodateien
 - Aufbau 34
 - bedingte Logik 76
 - Beispiel 35
 - Beschreibung 1
 - Blöcke 36
 - Darstellungsabschnitt 33
 - DEFINE-Block 36
 - Deklarationsabschnitt 33
 - entwickeln 33
 - FUNCTION-Block 37
 - Funktionen 54
 - Geltungsbereich für Kennungen 41
 - Generieren von HTML 67
 - HTML-Block 38
 - IF-Blöcke 76

Makrodateien (*Forts.*)

- Navigation innerhalb und zwischen 38
- permanente 101
- Schleifen 79
- Variablen 40
- WHILE-Blöcke 79

MESSAGE-Block

- Beispiel 59
- Beschreibung 58
- Geltungsbereich 58
- Syntax 58
- Verarbeitung 58

N

Navigation, innerhalb und zwischen Makros 38

Net.Data

- aufrufen 27
- Dateien, Zugriffsrechte 19
- konfigurieren 5
- Makrodateien, entwickeln 33
- Sicherheitsmechanismen 25
- Übersicht 1
- zusätzliche Informationen vii

Net.Data-Makro. Siehe Makrodateien. 1

Net.Data-Programmdateiobjekt

- in CGI-BIN-Bibliothek kopieren 6
- in mehrere Bibliotheken kopieren 6

O

OUT-Parameter 60

P

Permanente Makrodateien 101

Pfadanweisungen

- Aktualisierungsrichtlinien 13
- EXEC_PATH 14
- FFI_PATH 16
- INCLUDE_PATH 15
- Konfigurieren in der Initialisierungsdatei 13
- MACRO_PATH 13
- schützen, Datenbestände 25

Programmverbindungen (Links)

- Aufrufen von Net.Data 27, 31, 106
- in Web-Seiten zum Aufrufen von Net.Data 29

R

REPORT-Block

- Beschreibung 69
- Formatieren der Datenausgabe 69
- Geltungsbereich 41
- gespeicherte Prozeduren 91
- Kopf- und Fußzeilen 70

REPORT-Blöcke

- Einschränkungen 75
- gespeicherte Prozeduren 92
- Richtlinien 75

RETURN_CODE, Variable 58, 60

RETURNS, Parameter 61

ROW-Block, Geltungsbereich für Kennungen 42

S

Schleifen, WHILE-Blöcke 79

Schließungsvariable, Web-Register 12

Schützen, Datenbestände 21

Sicherheit

- Angeben von Zugriffsrechten 19
- Authentifizierung 24
- Berechtigung 25
- Firewall 21
- Net.Data-Mechanismen 25
- Netzwerkverschlüsselung 23
- Übersicht 21

Sprachumgebungen 79

- Beispiele 17
- Konfigurieren in der Initialisierungsdatei 17
- Konfigurieren von Anweisungen
- ENVIRONMENT 17
- Variablen 53

SQL

- Benennungsmoduskonfigurationsvariable 12
- Isolationskonfigurationsvariable 11

Standardberichte

- Angeben für gespeicherte Prozeduren 91, 92
- ausgeben 70

Starten von Net.Data 27

T

Tabellenvariablen 50

Tabellenverarbeitungsvariablen 51

Token-Größen 40

Transaktionsverarbeitung 101

U

Übergeben von Parametern, gespeicherte

Prozeduren 90

Umgebungsvariablen 45

URL-Adressen

- Aufrufen von Net.Data 27, 31, 106
- Definieren von Variablen 43

V

Variablen

- Arten von 40, 44
- ausführbar 46
- bedingt 45

Variablen (*Forts.*)

- Bericht 52
- Beschreibung 40
- definieren 42
- für Listen 49
- für Tabellen 50
- für Umgebung 45
- Geltungsbereich 41
- Konfiguration, Anweisungen
 - Beschreibung 10
 - Initialisierungsdatei 10
 - Schließen der Web-Register
(DTWR_CLOSE_REGISTRIES) 12
 - SQL-Benennungsmodus
(DTW_SQL_NAMING_MODE) 12
 - SQL-Isolation (DTW_SQL_ISOLATION) 11
- Sprachumgebung 53
- Tabellenverarbeitung 51
- Token-Größen 40
- verdeckt 48
- Verweisen auf 43
- zusätzliche 51
- Variablenverweise, Verarbeitungsreihenfolge 60
- Verarbeiten von Ergebnismengen, gespeicherte Prozeduren 90
- Verdeckte Variablen
 - schützen, Datenbestände 25
 - Variablennamen verdecken 48
- Verschlüsselung, Netzwerk 23
- Verweisen auf Variablen 43

W

- WHILE-Blöcke 79

Z

- Zugriffsrechte für Net.Data-Dateien angeben 19
- Zusätzliche Variablen 51

