Natural for Mainframes

Natural Optimizer Compiler

Version 9.2.1

November 2024

ADABAS & NATURAL

Dieses Dokument gilt für Natural for Mainframes ab Version 9.2.1.

Hierin enthaltene Beschreibungen unterliegen Änderungen und Ergänzungen, die in nachfolgenden Release Notes oder Neuausgaben bekanntgegeben werden.

Dokument-ID: ATMF-NOC-921-20241107DE

Inhaltsverzeichnis

Vorwort

Diese Dokumentation für den Natural Optimizer Compiler behandelt verschiedene Aspekte, die zu berücksichtigen sind, wenn der Natural Optimizer Compiler in Ihrer Umgebung installiert ist.

Anmerkungen:

1. Innerhalb dieser Dokumentation wird statt des Produktnamens „Natural Optimizer Compiler“ häufig der Produktcode „NOC“ verwendet.
2. Eine Erläuterung der in dieser Dokumentation benutzten Formatabkürzungen finden Sie im Abschnitt *Mögliche Formate* in der Natural *Statements*-Dokumentation.
3. Die Sprache der Benutzungsoberfläche des Produkts ist Englisch. Daher sind in den folgenden Dokumenten alle zur Orientierung erforderlichen englischen Literale beibehalten und ggf. nur in Klammern übersetzt worden.

Allgemeine Informationen	Verschiedene Aspekte des Natural Optimizer Compiler und wie Sie den größten Nutzen aus dem Natural Optimizer Compiler ziehen.
Optimizer Compiler benutzen	Statements und Programme, die für die Kompilierung verwendet werden. Statistikdaten zu Programmen, die sich für die Verarbeitung durch den Natural Optimizer Compiler eignen: Kommando `NOCSTAT`. Beispiele für den Einsatz des Optimizer Compiler.
Optimizer Compiler aktivieren	Wie der Natural Optimizer Compiler eingeschaltet wird.
Optimizer-Optionen	Verschiedene Optionen des Natural Optimizer Compiler. Anwendung von `PGEN` für die Ausgabe von generiertem Code und internen Natural Strukturen zur Untersuchung. Beeinflussung durch andere Natural-Parameter.
Performance-Überlegungen	Wie die beste Leistung unter Berücksichtigung von Datenformaten, Arrays, alphanumerischen Feldern, `DECIDE ON` und numerischen Werten erzielt wird.
Zaps auflisten	Übersicht über die Zaps, die beim Natural Optimizer Compiler angewendet wurden.

Verwandte Dokumentation:

Installing the Natural Optimizer Compiler on z/OS in der Natural *Installation*-Dokumentation

1 Über diese Dokumentation

Dokumentationskonventionen

Konvention	Beschreibung
Fettschrift	>Kennzeichnet Elemente auf einem Bildschirm.
`Nichtproportionale Schrift`	Kennzeichnet Namen und Orte von Diensten im Format *`Ordner.Unterordner.Dienst`*, Programmierschnittstellen (APIs), Namen von Klassen, Methoden und Properties in Java.
`Kursivschrift`	Kennzeichnet: Variablen, für die Sie situations- oder umgebungsspezifische Werte angeben müssen. Neue Begriffe, wenn sie erstmals im Text auftreten. Verweise auf andere Dokumentationsquellen.
`Nichtproportionale Schrift`	Kennzeichnet: Text, den Sie eingeben müssen. Meldungen, die vom System angezeigt werden. Programmcode.
{ }	Zeigt eine Reihe von Auswahlmöglichkeiten an, von denen Sie eine auswählen müssen. Geben Sie nur die innerhalb der geschweiften Klammern vorhandenen Informationen ein. Geben Sie nicht die Klammersymbole { } ein.
|	Trennt zwei sich gegenseitig ausschließende Auswahlmöglichkeiten in einer Syntaxzeile voneinander ab. Geben Sie eine der Auswahlmöglichkeiten ein. Geben Sie nicht das Symbol | ein.
[ ]	Zeigt eine oder mehrere Optionen an. Geben Sie nur die innerhalb der eckigen Klammern vorhandenen Informationen ein. Geben Sie nicht die Klammersymbole [ ] ein.
...	Zeigt an, dass Sie mehrere Auswahlmöglichkeiten desselben Typs eingeben können. Geben Sie nur die Informationen ein. Geben Sie nicht die drei Auslassungspunkte (...) ein.

Online-Informationen und Support

Produktdokumentation

Sie finden die Produktdokumentation auf unserer Dokumentationswebsite unter **https://documentation.softwareag.com**.

Zusätzlich können Sie auch über **https://www.softwareag.cloud** auf die Dokumentation für die Cloud-Produkte zugreifen. Navigieren Sie zum gewünschten Produkt und gehen Sie dann, je nach Produkt, zu „Developer Center“, „User Center“ oder „Documentation“.

Produktschulungen

Sie finden hilfreiches Produktschulungsmaterial auf unserem Lernportal unter **https://knowledge.softwareag.com**.

Tech Community

Auf der Website unserer Tech Community unter **https://techcommunity.softwareag.com** können Sie mit Experten der Software AG zusammenarbeiten. Von hier aus können Sie zum Beispiel:

- Unsere umfangreiche Wissensdatenbank durchsuchen.
- In unseren Diskussionsforen Fragen stellen und Antworten finden.
- Die neuesten Nachrichten und Ankündigungen der Software AG lesen.
- Unsere Communities erkunden.
- Unsere öffentlichen Repositories auf GitHub and Docker unter **https://github.com/softwareag** und **https://hub.docker.com/publishers/softwareag** besuchen und weitere Ressourcen der Software AG entdecken.

Produktsupport

Support für die Produkte der Software AG steht lizenzierten Kunden über unser Empower-Portal unter **https://empower.softwareag.com** zur Verfügung. Für viele Dienstleistungen auf diesem Portal benötigen Sie ein Konto. Wenn Sie noch keines haben, dann können Sie es unter **https://empower.softwareag.com/register** beantragen. Sobald Sie ein Konto haben, können Sie zum Beispiel:

- Produkte, Aktualisierungen und Programmkorrekturen herunterladen.
- Das Knowledge Center nach technischen Informationen und Tipps durchsuchen.
- Frühwarnungen und kritische Alarme abonnieren.
- Supportfälle öffnen und aktualisieren.
- Anfragen für neue Produktmerkmale einreichen.

Datenschutz

Die Produkte der Software AG stellen Funktionen zur Verarbeitung von personenbezogenen Daten gemäß der Datenschutz-Grundverordnung (DSGVO) der Europäischen Union zur Verfügung. Gegebenenfalls sind in der betreffenden Systemverwaltungsdokumentation entsprechende Schritte dokumentiert.

I Allgemeine Informationen

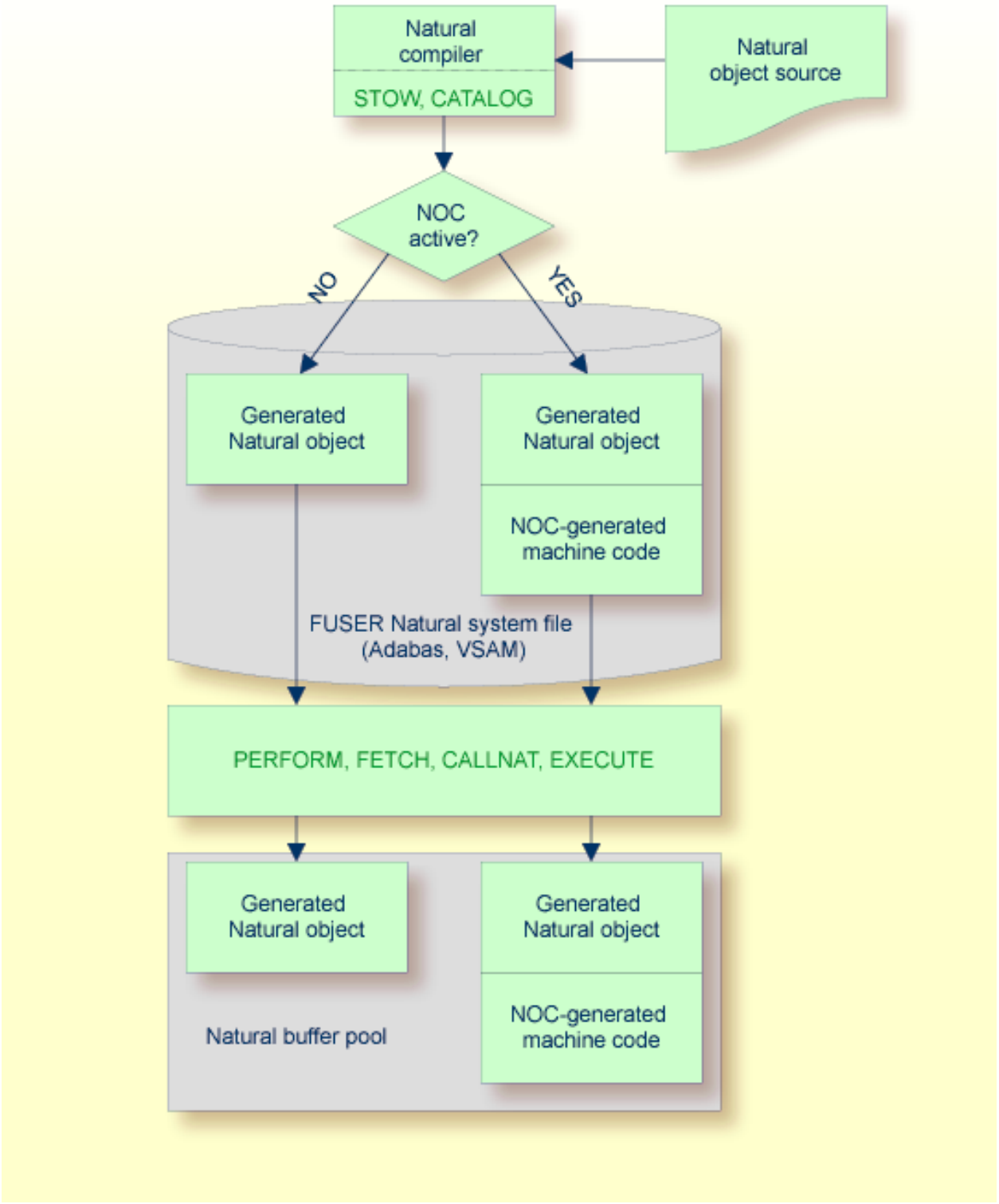
2 Allgemeine Informationen

Dieses Kapitel behandelt verschiedene Aspekte, die es zu berücksichtigen gilt, wenn der Natural Optimizer Compiler in Ihrer Umgebung installiert ist. Die Informationen in dieser Dokumentation helfen Ihnen, die Vorteile des Natural Optimizer Compiler voll auszuschöpfen.

Optimierung durch den Natural-Nukleus

Der Natural-Nukleus optimiert einfache Arithmetik-, Zuweisungs- und Vergleichs-Statements, indem er Teile von diesen in Maschinencode übersetzt. Alle Programme werden auf diese Weise automatisch optimiert.

Die folgende Grafik zeigt, wie der Natural Optimizer Compiler Maschinencode generiert, wenn ein Natural-Objekt kompiliert oder ausgeführt wird:


Natural compiler
STOW, CATALOG
Natural object source
NOC active?
NO
YES
Generated Natural object
Generated Natural object
NOC-generated machine code
FUSER Natural system file (Adabas, VSAM)
PERFORM, FETCH, CALLNAT, EXECUTE
Generated Natural object
Generated Natural object
NOC-generated machine code
Natural buffer pool

Optimierung durch den Natural Optimizer Compiler

Der Natural Optimizer Compiler geht einen Schritt weiter als die Standard-Optimierung. Er kompiliert nicht nur einfache Statements in Maschinencode, sondern auch komplexe Statements und Statement-Sequenzen.

Der kompilierte Code wird einer weitergehenden Optimierung hinsichtlich Array-Bereichsoperationen, Feldverkettung und optimale Basisregisterzuweisung unterzogen.

Alle mit dem Natural Optimizer Compiler optimierten Statements (inklusive Aritmetikoperationen) liefern gleiche Ergebnisse wie die mit Natural standardmäßig generierten Statements.

Zur Aktivierung des Natural Optimizer Compiler können Sie das Makro `NTOPT` im Natural-Parametermodul, den dynamischen Profilparameter `OPT`, das Systemkommando `NOCOPT` oder das Statement `OPTIONS` benutzen.

Alle Programme, die bei aktiviertem Natural Optimizer Compiler katalogisiert werden (mit Systemkommando `STOW` oder `CATALOG`), werden in Maschinencode kompiliert. Das hat auch zur Folge, dass die Objektcodegröße der Programme größer als üblich ist, je nachdem wie viel des Programms optimiert werden kann.

Ein mit dem Systemkommando `RUN` ausgeführtes Programm wird in Maschinencode kompiliert, wenn der Natural Optimizer Compiler mit dem Systemkommando `NOCOPT`, dem Makro `NTOPT` oder dem `OPTIONS`-Statement für das gesamte Programm oder einen Teil davon aktiviert ist.

Mit dem Kommando `NOCSTAT` können Sie feststellen, ob sich ein Programm für die Kompilierung mit dem Natural Optimizer Compiler eignet.

Anmerkung: Bei Programmen, die in Maschinencode kompiliert worden sind, kann die Einstellung `ON` des Profilparameters `RECAT` zum dynamischen Rekatalogisieren nicht benutzt werden.

Für das Ausführen von Programmen, die mit dem Natural Optimizer Compiler kompiliert wurden, braucht der Natural Optimizer Compiler nicht installiert zu sein.

II Optimizer Compiler benutzen - Übersicht

3 Was kompiliert wird und was nicht

Der Natural Optimizer Compiler ist besonders wirkungsvoll bei Programmen, die eine beträchtliche Menge an Datenmanipulationen enthalten, wie z. B. Berechnung, Transfer und Verarbeitung logischer Bedingungen.

Dieser Abschnitt gibt eine Übersicht über die Statements, die in Maschinencode kompiliert werden, und über die Statements, die nicht kompiliert werden.

Anmerkung: Die Optionen, die der Natural Optimizer Compiler zur Verfügung stellt, können nicht verwendet werden, um Statements festzulegen, die optimiert werden sollen, wie in den *Optimizer-Optionen* beschrieben.

Statements, die vom Natural Optimizer Compiler kompiliert werden

Der Natural Optimizer Compiler kompiliert die folgenden Statements in Maschinencode:

- **Statements für arithmetische und Datenverschiebungs-Operationen:**
  - `ADD`
  - `ASSIGN`
  - `COMPRESS`
  - `COMPUTE`
  - `DIVIDE`
  - `EXAMINE`, mit folgenden Klauseln:
    - `DIRECTION` (nur mit Konstantenwerten, d.h. `FORWARD` oder `BACKWARD`),
    - `GIVING NUMBER`, `GIVING POSITION` (auch gleichzeitig),
    - `GIVING LENGTH`
    - `EXAMINE .. FOR <fld1> REPLACE <fld2>`, wenn das `EXAMINE`-Feld den Typ (Alpha/Binär) hat und die Search/Replace-Felder die gleiche Länge im Bereich (1:256) haben.
    - `TRANSLATE INTO (UPPER/LOWER) CASE`

    Beispiel:

    ```
    EXAMINE #TEXT FOR #A GIVING NUMBER #NMB1
    EXAMINE #TEXT FOR #A GIVING POSITION #POSEX5
    EXAMINE #TEXT FOR #A GIVING LENGTH #LGHEX6 ↩
    ```

    Einschränkungen:

    - `GIVING INDEX` wird nicht optimiert.
    - *operand1* und *operand4* können feste Array-Ausprägungen sein, d.h. es sind keine Bereiche zulässig, zum Beispiel:

```
EXAMINE #A(#J) FOR #B(#K)
```

- MOVE (ROUNDED, SUBSTRING, BY NAME, LEFT/RIGHT JUSTIFIED,)
- MOVE ALL
- MULTIPLY
- RESET
- SUBTRACT

- **Statements für die Verarbeitung von logischen Bedingungen:**
  - IF
  - DECIDE FOR
  - DECIDE ON
- **Statements für die Ausführung von Schleifen:**

  FOR
- ESCAPE
- REPEAT

Statements, die nicht kompiliert werden

Der Natural Optimizer Compiler kompiliert *nicht* die folgenden Statements:

- Ein-/Ausgabe-(I/O-)Statements (DISPLAY, WRITE, READ/WRITE WORK FILE).
- Komplexe spezielle Statements, zum Beispiel SEPARATE.
- Statements, die die Steuerung an ein anderes Objekt übergeben, zum Beispiel FETCH, PERFORM, CALLNAT, CALL.
- Statements, die einen Datenbankzugriff ausführen (READ, FIND, HISTOGRAM, GET, UPDATE, DELETE, END TRANSACTION, BACKOUT TRANSACTION)

4 NOCSTAT-Kommando

Bei Programmen, die mit dem Natural Optimizer Compiler optimiert wurden, können bestimmte Statements beim Katalogisieren direkt in Maschinencode umgewandelt werden. Dadurch kann beim Ausführen der optimierten Objekte mit Natural zur Laufzeit die Leistung erheblich verbessert werden.

Das NOCSTAT-Kommando analysiert katalogisierte Objekte und liefert statistische Informationen, die bei der Entscheidung helfen, ob Programm-Statements von der Optimierung durch den Natural Optimizer Compiler profitieren und, wenn ja, in welchem Umfang sie optimiert werden können.

Wenn ein Programm katalogisiert wird (mit STOW, CATALL), erzeugt der Natural Optimizer Compiler einen internen (Pseudo-)Objektcode basierend auf den Statements des Source-Programms. In den meisten Fällen wird ein Statement des Source-Programms in ein Statement der Pseudocode-Instruktion umgewandelt. Für komplexe Statements, wie FOR und REPEAT, werden jedoch mehrere Pseudocode-Instruktionen generiert. Die NOCSTAT-Analysen basieren auf den generierten Pseudocode-Instruktionen. Daher kann die Anzahl der in den statistischen Reports aufgeführten Statements die Anzahl der Statements im Source-Programm übersteigen.

NOCSTAT aufrufen

Um das Natural-Kommando NOCSTAT zu benutzen:

- Geben Sie das Direktkommando NOCSTAT ein.

  Der NOCSTAT-Hauptbildschirm wird angezeigt (Beispiel):

```
16:41:00            ***** NATURAL NOCSTAT COMMAND *****            2017-11-24


Name ................. ________
Library ............... SAGTEST_

NOCable Objects only .. _

Output Report ......... X Statement Category
                        _ Statement Type
                        _ Code Profile

Output Destination .... X Screen
                        _ CSV to Work File
                        _ XML to Work File
                          with XSL ____________________________________

Progress Control ...... X
Download to PC ........ _


Command ===>


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help        Exit                                                 Canc
```

Um Hilfe zu Feldern zu erhalten, können Sie entweder ein Fragezeichen (?) in das betreffende Feld eingeben und ENTER drücken oder den Cursor in das Feld stellen und PF1 drücken. Drücken Sie PF3, um `NOCSTAT` zu beenden.

Reports generieren

Sie können statistische Reports für ein einzelnes Programm oder eine Gruppe von Programmen generieren. Wenn Sie mehr als ein Programm auf einmal analysieren, werden die Reports in Serie erstellt. Wenn Sie einen Report fertig angesehen haben, drücken Sie ENTER, um den nächsten Report anzuzeigen.

Der Bildschirm `NATURAL NOCSTAT COMMAND` bietet die folgenden Optionen:

Feld	Erklärung	
Name	Geben Sie einen Namen oder einen Namensbereich ein, um das oder die Programme anzugeben, die Sie prüfen möchten:	
	value ist ein beliebiges einzelnes Zeichen oder eine Kombination aus mehreren Zeichen.	
	value	Einzelnes Programm.
	*	Alle Programme.
	*value**	Alle Programme, deren Namen mit *value* beginnen.
	value>	Alle Programme, deren Namen größer gleich *value* sind.
	value<	Alle Programme, deren Namen kleiner gleich *value* sind.
Library	Geben Sie den Namen einer Library oder einen Namensbereich ein: Weitere Eingaben wie zuvor beim Feld Name beschrieben. Standardwert ist die aktuelle Library.	
NOCable Objects only	Markieren Sie diese Option, um Programme auszuschließen, die bereits mit dem Natural Optimizer Compiler kompiliert sind. Andernfalls wählt das Kommando NOCSTAT standardmäßig alle Natural-Programme aus, die in den Feldern Name und Library angegeben sind, inklusive der mit NOC kompilierten Programme.	
Output Report	Markieren Sie eine der Optionen, um Statements nach Kategorie, Typ oder Codeprofil auszuwählen. Siehe ***Statement-Kategorie***, ***Statement-Typ*** und ***Codeprofil*** weiter unten.	
Output Destination	Markieren Sie eine der Optionen, um das Format und den Zielort der Ausgabe festzulegen:	
	Screen	Der Report wird im Bildschirm angezeigt oder die Reportdaten werden in die Druckdatei (Print File) 7 geschrieben, wenn Download to PC für die Verarbeitung gewählt ist.
	CSV to Work File	Erzeugt Tabellenkalkulationen (Speadsheets) mit kommagetrennten Werten. Die Reportdaten werden in eine der folgenden Dateien geschrieben: 1. Arbeitsdatei (Work File) 7 bei Online-Betrieb und gewählter Option und Download to PC. 2. Arbeitsdatei (Work File) 1 in allen anderen Fällen. Benutzen Sie die Dateinamenserweiterung .csv, um die Arbeitsdatei zur Weiterverarbeitung direkt auf Ihren PC zu schreiben. Um Reports an einen PC weiterleiten zu können, muss Entire Connection installiert sein.

Feld	Erklärung	

	XML to Work File	Generiert XML-Dokumente.
		
		Die Reportdaten werden in eine der folgenden Dateien geschrieben:
		
		1. Arbeitsdatei (Work File) 7 im Online-Betrieb und bei gewählter Option Download to PC.
		2. Arbeitsdatei (Work File) 1 in allen anderen Fällen.
		
		Benutzen Sie die Dateinamenserweiterung .txt, um die Arbeitsdatei direkt auf Ihren PC zu schreiben, und ändern Sie die Dateinamenserweiterung anschließend in .xml für die Weiterverarbeitung.
		
		Um Reports an einen PC weiterleiten zu können, muss Entire Connection installiert sein.
		
		Wird im Feld with XSL ein Wert eingegeben, dann wird oben im XML-Ausgabedokument eine Verarbeitungsanweisung hinzugefügt:
		
		`<?xml-stylesheet type="text/xsl" href="`
		`value "?>`
		
		Der eingegebenen Wert *value* sollte die absolute oder relative URL des Style Sheet sein, z.B.:
		
		`nocstat.xsl`
		
		oder
		
		`http://natural.software-ag.de/nocstat.xsl`
		
		Die Verarbeitungsanweisung bewirkt, dass das Dokument gemäß dem angegebenen Style Sheet umgeformt wird, wenn es mit einem XSLT-fähigen Browser betrachtet oder in einem XSLT-Batch-Lauf umgeformt wird. Eine typische Verwendung dieser Funktion ist die Umwandlung des ausgegebenen XML in eine HTML-Seite.
		
		Zusammen mit Natural werden zwei XSLT Style Sheets als Textobjekte ausgeliefert: NOCSTLS1 und NOCSTLS2, beide in der Natural Library SYSEXUEX in der Systemdatei FNAT.
		
		Wie weiter unten beschrieben, enthält NOCSTLS1 Formatierungsanweisungen für den Report-Typ ***Statement-Kategorie*** und NOCSTLS2 für den Report-Typ ***Statement-Typ***.

Feld	Erklärung	
		Laden Sie die Style Sheets mit der Dateinamenserweiterung `.xsl` in dasselbe Verzeichnis, in dem die XML-Arbeitsdateien gespeichert werden.
`Progress Control`	Gilt nur in einer Online-Umgebung und wenn eine der folgenden Optionen für die Verarbeitung gewählt wird: 1. `CSV to Work File`, 2. `XML to Work File`, 3. `Download to PC`. Wird eine dieser Optionen gewählt, erschient bei jedem im generierten Report aufgelisteten Programm eine kurze Meldung.	
`Download to PC`	Gilt nur, wenn Entire Connection installiert ist und wenn Sie im Online-Betrieb mit Druck-/Arbeitsdatei (Print/Work File) 7 arbeiten und diese als die PC-Datei definiert ist (siehe Parameter `WORK` und `PRINT`). Zum Herunterladen der Reportausgabe auf einen PC mittels Entire Connection können Sie benutzen: 1. Druckdatei (Print File) 7 für das Ausgabeziel Bildschirm (`Screen`). 2. Arbeitsdatei (Work File) 7 für die Ausgabe im Format CSV oder XML.	

Report-Formate

Sie können zwischen drei nachfolgend beschriebenen Ausgabeformaten wählen, um die Statistiken anzuzeigen, die `NOCSTAT` für die analysierten Statements liefert.

Für Programme, die bereits mit dem Natural Optimizer Compiler optimiert wurden, und für Programme, die für eine Optimierung in Frage kommen, werden unterschiedliche Report-Layouts erstellt. Die unten stehenden Beispiel-Reports zeigen den Unterschied.

Drücken Sie PF3, um die Report-Bearbeitung zu unterbrechen und zum Bildschirm `NATURAL NOCSTAT COMMAND` zurückzukehren.

Nachfolgend finden Sie Informationen zu:

- Statement-Kategorie
- Statement-Typ

- Codeprofil

Statement-Kategorie

Der mit der Option `Statement Category` generierte Statistik-Report listet verschiedene Kategorien von Statements mit der entsprechenden Anzahl der Ausprägungen und die Gesamtzahl der bereits optimierten oder zur Optimierung geeigneten Statements, je nachdem, ob das Programm mit dem Natural Optimizer Compiler optimiert wurde oder nicht.

Beispiel für ein Programm ohne NOC-Optierung:

```
11:49:46             ***** NATURAL NOCSTAT COMMAND *****             2017-05-29
                Library SAGTEST  Name NOCTEST1 Type Program


                     No NOC      NOCable
                    --------     --------
      Database Loop:        0            0
    Database Simple:        0            0
    SORT / WORK I/O:        0            0
       FOR / REPEAT:        0            1
   Screen / Printer:        1            0
String Manipulation:        6           34
    Arith / Logical:        0          996
      Program Calls:       20            0
   Control Transfer:        2          182
        Block Start:        1            0
    Set Environment:        7            0
   System Functions:        2            0
      Miscellaneous:        0            1

               Total Statements:     1254
                NOC optimizable:     1214  ( Ratio:  96 % )
                Longest NOC Run:      216  Statements
```

Beispiel für ein mit NOC optimiertes Programm:

```
11:51:25             ***** NATURAL NOCSTAT COMMAND *****             2017-05-29
                Library SAGTEST  Name NOCTEST1 Type Program

     MCG Options: (ON,OVFLW,INDX,MIX,IO)


      Database Loop:        0
    Database Simple:        0
    SORT / WORK I/O:        0
       FOR / REPEAT:        0
   Screen / Printer:        1
String Manipulation:       36
    Arith / Logical:        0
      Program Calls:       20
   Control Transfer:        2
        Block Start:        1
     Set Environment:        7
   System Functions:        2
      Miscellaneous:        1

   Total Statements:     1255
      NOC optimized:     1185  ( Ratio:  94 % )
    Longest NOC Run:      136  Statements
```

Spalten und Felder im Report:

Spalte	Erklärung
`No NOC`	Nicht zur Optimierung mit NOC geeignete Statements.
`NOCable`	Zur Optimierung mit NOC geeignete Statements: **Anmerkung:** Die Anzahl der mit NOC optimierbaren (`NOCable`) Statements ist nur eine vernünftige Annahme, kann aber nicht als absolut zuverlässiger Wert betrachtet werden. Dies liegt daran, dass das `NOCSTAT`-Kommando nicht alle analytischen Abfragen und gelegentlich auch sehr komplexe Code-Untersuchungen durchführen kann, die definitiv entscheiden, ob ein Statement mit dem Natural Optimizer Compiler optimiert werden kann.
Feld	
`Database Loop`	Die Anzahl der Datenbank-Statements, die eine Verarbeitungsschleife erzeugen, z.B. `FIND` und `READ`.
`Database Simple`	Datenbank-Statements, die keine Verarbeitungsschleife erzeugen, z.B. `STORE`, `UPDATE`, `DELETE` und `GET`.
`SORT / WORK I/O`	`SORT` und `WORKFILE`-Statements.
`FOR / REPEAT`	Statements, die Verarbeitungsschleifen erzeugen.
`Screen / Printer`	Statements für Bildschirm- und Drucker-Ein-/Ausgaben, z.B. `WRITE`, `DISPLAY` und `INPUT`.

Spalte	Erklärung
String Manipulation	Statements zur Zeichenketten-Manipulation, z.B. EXAMINE und COMPRESS.
Arith / Logical	Arithmetische und logische Statements, z.B. MOVE, COMPUTE und IF.
Program Calls	Statements zur Übergabe der Steuerung an eine Subroutine oder ein Subprogramm, z.B. PERFORM, CALLNAT und FETCH.
Control Transfer	Statements zur Durchführung von Sprüngen innerhalb des Programms, z.B. ESCAPE BOTTOM, FOR und REPEAT Schleifen.
Block Start	Nicht-ausgeführte Statements, die Code-Blöcke abgrenzen, z.B. DEFINE SUBROUTINE und AT END. Diese Statements werden niemals optimiert, weil sie nie ausgeführt werden.
Set Environment	Statements zum Einstellen der Umgebung, z.B. SET CONTROL, SET GLOBALS und SET KEY.
System Functions	Statements wie zum Beispiel TOTAL, SUM, COUNT, MAX, MIN und *COUNT.
Miscellaneous	Pseudo-Code-Statements, die für die Optimierung nicht relevant sind und deshalb vom Natural Optimier Compiler ignoriert werden.
Gesamtzahlen	
Total Statements	Gesamtzahl der im Programm gefundenen Statements. Es ist möglich, dass diese Zahl nicht der Zahl der tatsächlich im Sourcecode befindlichen Statements entspricht (wie schon in der Einleitung zum NOCSTAT-Kommando weiter oben beschrieben).
NOC optimized	Bei einem optimierten Programm sind dies die tatsächlichen Pseudo-Code-Statements (wie schon in der Einleitung zum NOCSTAT-Kommando weiter oben beschrieben), die NOC-optimiert in Maschinencode kompiliert worden sind.
NOC optimizable	Bei nicht optimierten Programmen ist dies die mögliche Zahl der Statements, die optimiert werden konnten. Die Zahl kann etwas höher sein als die tatsächliche Zahl, da bestimmte Faktoren im NOCSTAT-Programm nicht berücksichtigt werden. So kann zum Beispiel ein SUBSTRING-Statement, das mehr als vier Ausprägungen hat, als „optimierbar“ angezeigt werden, obwohl es nicht optimiert wird.
Ratio	Verhältnis zwischen Total Statements und NOC-optimized Statements oder Total Statements und NOC-optimizable statements in Prozent.
Longest NOC Run	Mit NOC optimiertes Programm: Die Anzahl an zusammenhängenden optimierten Statements - je weniger Fragment-Sequenzen, desto besser die Leistung.
	Nicht mit NOC optimiertes Programm: Die Anzahl an zu erwartenden zusammenhängenden Statements, wenn das Programm optimiert wäre.

Statement-Typ

Der mit der Option `Statement Type` generierte Statistik-Report listet einzelne Statements mit der entsprechenden Anzahl der Ausprägungen und die für optimierte Objekte generierte NOC-Kodierung.

Beispiel für ein Programm ohne NOC-Optierung:

```
12:29:23            ***** NATURAL NOCSTAT COMMAND *****            2017-05-29
               Library SAGTEST  Name NOCTEST1 Type Program

       Statement            No NOC   NOCable
------------------------   --------  --------
MOVE/COMPUTE/ASSIGN               0       615
EXAMINE                           6         0
SEPARATE                          0        30
COMPRESS                          0         4
MOVE TO SYSTEM FUNCTION           2         0
CALLNAT/PERFORM EXTERNAL         17         0
MOVE EDITED                       1         0
ELSE/CLOSE LOOP                   0       182
ON ERROR                          1         0
END                               1         0
STOP                              1         0
IF                                0        51
IF IN REPEAT UNTIL                0         1
REPEAT                            0         1
RESET                             0        74
IF                                0       255
FETCH                             3         0
IGNORE                            0         1
STACK TOP CMD/DATA                2         0
MCG OPTIONS                       1         0
OPTIONS                           1         0
SET CONTROL                       4         0
```

Beispiel für ein mit NOC optimiertes Programm:

```
12:31:30            ***** NATURAL NOCSTAT COMMAND *****            2017-05-29
               Library SAGTEST  Name NOCTEST1 Type Program
     MCG Options: (ON,OVFLW,INDX,MIX,IO)
       Statement            Number
------------------------   --------
EXAMINE                          6
SEPARATE                        30
MOVE TO SYSTEM FUNCTION          2
CALLNAT/PERFORM EXTERNAL        17
MOVE EDITED                      1
NOC CODE                      1183
ON ERROR                         1
END                              1
STOP                             1
FETCH                            3
IGNORE                           1
STACK TOP CMD/DATA               2
MCG OPTIONS                      2
OPTIONS                          1
SET CONTROL                      4
```

Codeprofil

Der mit der Option `Code profile` generierte Statistik-Report zeigt zusammenhängende Sequenzen von Statements, gruppiert nach Kategorien in einem Source-Programm, das für die Optimierung geeignet ist, oder listet die für ein optimiertes Programm erzeugte NOC-Kodierung. Ausprägungen werden hervorgehoben angezeigt.

Beispiel für ein Programm ohne NOC-Optierung:

```
12:38:52            ***** NATURAL NOCSTAT COMMAND *****            2017-05-29
               Library SAGTEST  Name NOCTEST1 Type Program

Line   Statement
-----  -------------------------
0000   ON ERROR
0000   MCG OPTIONS
0000   OPTIONS
0295   CALLNAT/PERFORM EXTERNAL
0295   MOVE/COMPUTE/ASSIGN       <-- NOCable
0295   MOVE/COMPUTE/ASSIGN       <-- NOCable
0295   MOVE/COMPUTE/ASSIGN       <-- NOCable
0295   MOVE/COMPUTE/ASSIGN       <-- NOCable
0740   MOVE/COMPUTE/ASSIGN       <-- NOCable
0745   IF                        <-- NOCable
0750   MOVE/COMPUTE/ASSIGN       <-- NOCable
0755   MOVE/COMPUTE/ASSIGN       <-- NOCable
0760   CALLNAT/PERFORM EXTERNAL
0765   IF                        <-- NOCable
0770   MOVE/COMPUTE/ASSIGN       <-- NOCable
0775   ELSE                      <-- NOCable
0780   MOVE/COMPUTE/ASSIGN       <-- NOCable
0810   RESET                     <-- NOCable
MORE
```

Beispiel für ein mit NOC optimiertes Programm:

```
12:39:47            ***** NATURAL NOCSTAT COMMAND *****            2017-05-29
               Library SAGTEST  Name NOCTEST1 Type Program

Line   Statement
-----  -------------------------
0000   MCG OPTIONS
0005   MCG OPTIONS
0000   OPTIONS
0295   CALLNAT/PERFORM EXTERNAL
0295   NOC CODE
0295   NOC CODE
0295   NOC CODE
0295   NOC CODE
0740   NOC CODE
0745   NOC CODE
0750   NOC CODE
0755   NOC CODE
0760   CALLNAT/PERFORM EXTERNAL
0765   NOC CODE
0770   NOC CODE
0775   NOC CODE
0780   NOC CODE
0810   NOC CODE
MORE
```

Batch-Ausführung

Nachfolgend finden Sie Job-Beispiele für die Verarbeitung von NOCSTAT-Reports im Batch-Modus, um eine CSV-Arbeitsdatei zu erstellen. Nach erfolgter Job-Ausführung können die erzeugten Arbeitsdateien mit Standard-Transfer-Tools vom Host-Rechner auf einen PC zur Weiterverarbeitung übertragen werden.

Beispiel-Job :

```
//NOCBATCH JOB  (NOC,,,30),CLASS=K,MSGCLASS=X                          00000100
//NATEX EXEC PGM=NATvrsBA,REGION=6200K,PARM=('IM=D')                   00000200
//STEPLIB  DD DISP=SHR,DSN=TESTNAT.LOAD                                00000300
//CMPRINT  DD SYSOUT=X                                                 00000400
//CMWKF01  DD DSN='NOC.NOCSTAT.OUT',DISP=(NEW,CATLG),                  00000500
            SPACE=(CYL,(1,1)),UNIT=SYSDA,VOL=SER=SAG001                00000600
//SYSOUT   DD SYSOUT=X                                                 00000700
//CMSYNIN  DD *                                                        00000800
NOCSTAT                                                              00000900
*,library,,X,,,,X                                                    00001000
.                                                                    00001100
```

```
  FIN                                                                   00001200
  /*                                                                    00001300
```

5 Größe des Maschinencodes anzeigen

Mit dem Systemkommando `LIST DIRECTORY` können Sie sich anzeigen lassen, ob ein Programm in Maschinencode kompiliert worden ist und welche Größe der Maschinencode hat.

Um ein kompiliertes Programm aufzulisten:

- Geben Sie das folgende Natural-Systemkommando ein:

```
LIST DIR object-name
```

Das Kommando zeigt die Verzeichnisinformationen zu dem angegebenen Objekt an. Am unteren Rand des Bildschirms sehen Sie die Größe des NOC-Codes (vom Natural Optimizer Compiler generierter Maschinencode), die zum Kompilieren benutzten `OPT`-Parameter und die Version des Natural Optimizer Compiler, unter der das Programm katalogisiert wurde.

Weitere Informationen zum Kommando siehe `LIST` in der *Systemkommandos*-Dokumentation.

6 Beispiele für die Optimizer-Verwendung

Die folgenden Beispiele zeigen, wann der Natural Optimizer Compiler am besten eingesetzt werden kann, und geben einen Hinweis auf seine Leistungsfähigkeit:

Beispiel 1 - Keine Verbesserung

Es wäre nichts gewonnen, wenn Sie den Natural Optimizer Compiler für das folgende Programm benutzen, da es ein Statement, das einen Datenbankzugriff durchführt, und ein Ein-/Ausgabe-Statement enthält (siehe *Statements die nicht kompiliert werden*):

```
DEFINE DATA LOCAL
  1 EMPLOYEES VIEW OF EMPLOYEES
    2 JOB-TITLE
    2 BIRTH
    2 NAME
  END-DEFINE
  FIND EMPLOYEES WITH JOB-TITLE = 'PROGRAMMER' OR = 'ANALYST'
                              OR = 'PROGRAMMER/ANALYST'
                              OR = 'SYSTEM ANALYST'
    DISPLAY JOB-TITLE BIRTH NAME
  END-FIND
  END
```

Beispiel 2 - Beträchtliche Verbesserung

Wird das folgende Programm mit dem Natural Optimizer Compiler kompiliert, dann sieht man eine Leistungsverbesserung von ca. 30 % (das bedeutet eine Verringerung der CPU-Belastung um 30 %). Das gezeigte Beispiel-Programm führt eine statistische Analyse des Alters von IT-Mitarbeitern durch. Optimierte Statements sind in Fettschrift angegeben.

In diesem Beispiel erhöht der Natural Optimizer Compiler die Größe des Objekts um 20,5 % aufgrund von 952 Bytes an zusätzlichem Maschinencode.

Profilparameter-Einstellung	Buffer Pool-Größe	Größe des von NOC generierten Maschinencodes
OPT=NODBG	5768	952
OPT=OFF	4784	0

```
 DEFINE DATA
 LOCAL
 1 EMPLOY VIEW OF EMPLOYEES
   2 JOB-TITLE     (A25)
   2 BIRTH         (D)
 1 I               (I1)  INIT <1>
 1 CDATE           (D)
 1 NUMB            (N4)
 1 SUMM            (P7.2)
 1 SQUARE          (F8)
 1 DEVI            (F8)
 1 DEVIATION       (N3.4)
 1 MEAN            (P2.3)
 1 AGEDIS          (F8/1:70)
 1 AGEMAX          (F8)
 1 AGEH            (P3)
 1 AGE             (P3)
 1 AGEDAYS         (P15)
 1 LINE            (A71/1:20)
 1 REDEFINE LINE
   2 POINTS        (A1/1:20,0:70)
 END-DEFINE
 *
 MOVE *DATX TO CDATE
 *
 FIND EMPLOY WITH JOB-TITLE = 'PROGRAMMER' OR = 'ANALYST'
   OR = 'PROGRAMMER/ANALYST' OR = 'SYSTEM ANALYST'
AGEDAYS:= CDATE - BIRTH
   AGE:=AGEDAYS / 365
   ADD 1 TO AGEDIS(AGE)                  /* DISTRIBUTION
   ADD 1 TO NUMB
   ADD AGE TO SUMM
   COMPUTE SQUARE = SQUARE + AGE * AGE
 END-FIND
 *
 ***********************************************************************
 * COMPUTE ESTIMATES
 ***********************************************************************
 *
 COMPUTE DEVI = NUMB * SQUARE / (SUMM * SUMM) - 1
 COMPUTE DEVIATION = SQRT(DEVI)
 COMPUTE MEAN = SUMM / NUMB
 *
 ***********************************************************************
 * GRAPHIC DISPLAY
 ***********************************************************************
 *
 FOR I 1 70
   IF AGEDIS(I) > AGEMAX MOVE AGEDIS(I) TO AGEMAX
   END-IF
 END-FOR
```

```
FOR I 1 70
  COMPUTE AGEDIS(I) = AGEDIS(I) * 20 / AGEMAX
END-FOR
FOR I 1 70
  COMPUTE AGEH = 21 - AGEDIS(I)
  IF AGEH < 21 MOVE '*' TO POINTS(AGEH:20,I)
  END-IF
END-FOR
*
************************************************************************
* COMPLETE GRAPHIC DISPLAY
************************************************************************
*
MOVE '!' TO POINTS(*,0)
WRITE TITLE LEFT
 AGEMAX(EM=999)  20X  'DISTRIBUTION OF IT-EMPLOYEES BY AGE'
WRITE NOTITLE NOHDR
LINE(*) /
'0--------10--------20--------30--------40--------50--------60--------'
/  'MEAN='
```

Beispiele 3 und 4 - CPU-Nutzung

Das folgende Programm veranschaulicht den Unterschied in der CPU-Auslastung in Abhängigkeit von den Optionen, die Sie beim Kompilieren des Programms auswählen. Die folgende Tabelle listet die CPU-Auslastung in Sekunden und Prozent auf. Die in der Tabelle angegebenen Zahlen wurden während eines Testlaufs in einer IBM z/OS-Umgebung ermittelt. Sie können nur als allgemeine Orientierung dienen, da die absoluten Werte je nach eingesetzter Hardware variieren.

```
DEFINE DATA LOCAL
1 #I1         (I4) INIT <1>
1 #I2         (I4) INIT <2>
1 #J1         (I4) INIT <3>
1 #J2         (I4) INIT <4>
1 #F          (I4)
1 #ARR1       (N7/10,5)
1 #ARR2       (N5/10,5)
END-DEFINE
*
FOR #F = 1 TO 1000000
  MOVE #ARR1(#I1,#I2) TO #ARR2(#J1,#J2)
END-FOR
*
END
```

Option	CPU Sekunden	CPU Prozentsatz
OFF	8.78	100
ON	0.63	7.18
INDX	0.85	9.68
OVFLW	1.71	19.48
INDX,OVFLW	2.00	22.78
INDX,OVFLW,NODBG	1.61	18.34
INDX,OVFLW,NODBG,NOSGNTR	1.61	18.34
NODBG	0.44	5.01
NOSGNTR	0.63	7.18
NODBG,NOSGNTR	0.44	5.01

```
DEFINE DATA LOCAL
1 #I1         (P7) INIT <1>
1 #I2         (P7) INIT <2>
1 #J1         (N7) INIT <3>
1 #J2         (N7) INIT <4>
1 #K1         (I4) INIT <5>
1 #K2         (I4) INIT <6>
1 #F          (I4)
1 #FIELD1     (P5)
1 #FIELD2     (N5)
1 #FIELD3     (I2)
END-DEFINE
*
FOR #F = 1 TO 500000
*
  #FIELD1:= #I1 - #I2 + (13 * 10 / 5)
  #FIELD2:= #J1 - #J2 + (13 * 10 / 5)
  #FIELD3:= #K1 - #K2 + (13 * 10 / 5)
*
END-FOR
*
END
```

Option	CPU Sekunden	CPU Prozentsatz
OFF	18.61	100.00
ON	4.95	26.60
INDX	4.95	26.60
OVFLW	5.38	28.91
INDX,OVFLW	5.38	28.91
INDX,OVFLW,NODBG	5.26	28.26
INDX,OVFLW,NODBG,NOSGNTR	5.09	27.35

Option	CPU Sekunden	CPU Prozentsatz
NODBG	4.79	25.74
NOSGNTR	4.81	25.85
NODBG,NOSGNTR	4.63	24.88
NODBG,NOSGNTR,ZD=OFF	4.51	24.23
NODBG,NOSGNTR,ZD=OFF,SIGNCHCK=OFF	4.41	23.70

III

7 Optimizer Compiler aktivieren

Um den Natural Optimizer Compiler zu aktivieren, können Sie eine der in den folgenden Abschnitten beschriebenen Methoden verwenden, wobei die erste Alternative die statischste und die letzte Alternative die dynamischste ist.

Bei allen Alternativen werden die Optimizer-Optionen wie im Abschnitt *Optimizer-Optionen* beschrieben verwendet. Durch Verwendung dieser Optionen können Sie steuern, wie und wann Maschinencode generiert wird, welche Trace-Optionen verwendet werden sollen und was die Zielarchitektur sein wird. Die Optimizer-Optionen sind der einzige Steuermechanismus für den Natural Optimizer Compiler.

Makro NTOPT

Mit dem Makro `NTOPT` im Natural-Parametermodul können Sie den Natural Optimizer Compiler statisch für einen verlinkten Natural-Nukleus aktivieren. Immer wenn dieser Natural-Nukleus gestartet wird, werden wieder die gleichen Optimizer-Optionen verwendet.

Beispiel 1:

```
        NTOPT 'INDX,OVFLW,ZD=OFF'
```

Beispiel 2:

```
        NTOPT 'INDX,OVFLW,ZD=OFF,TRGPT',                                *
              'TRSTMT,OPTLEV03'
```

Beachten Sie bitte den Stern „*“ als Fortsetzungszeichen in Spalte 72.

Eine Erklärung der verwendeten Optionseinstellungen finden Sie im Abschnitt *Optimizer-Optionen*.

Dynamischer Profilparameter OPT

Wenn Sie eine Natural-Sitzung starten, können Sie den Natural Optimizer Compiler dynamisch aktivieren, indem Sie den Natural-Profilparameter `OPT` angeben. Als Synonym für `OPT` können Sie `MCG` angeben. Die Angabe im Parametermodul wird überschrieben. Die Optionen sind nur für die aktuelle Sitzung gültig.

Beispiel:

```
OPT=(INDX,OVFLW,ZD=OFF)
```

oder

```
MCG=(INDX,OVFLW,ZD=OFF)
```

Eine Erklärung der verwendeten Optionseinstellungen finden Sie im Abschnitt *Optimizer-Optionen*.

Systemkommando NOCOPT

Wenn Sie eine Natural-Sitzung gestartet haben, können Sie den Natural Optimizer Compiler mit dem Natural-Systemkommando `NOCOPT` aufrufen. Der Bildschirm zeigt die aktuelle Einstellung der Natural Optimizer Compiler-Optionen, so wie sie während des Natural-Starts angegeben wurden. Sie können die Einstellungen im Online-Betrieb ändern.

Die geänderten Parametereinstellungen sind nur für die aktuelle Sitzung gültig.

Natural-Statement OPTIONS

Der Parameter `MCG` des Natural-Compiler-Statement `OPTIONS` ermöglicht die flexibelste und leistungsfähigste Steuerung der Maschinencodegenerierung, weil es möglich ist, verschiedene Optionen für einzelne Statements in einem Programm zu setzen. Auf diese Weise kann man innerhalb eines einzelnen Programmes den Natural Optimizer Compiler mehrere Male aktivieren und deaktivieren, um Bereiche von Statements mit verschiedenen Optionseinstellungen einzuschließen.

Beispiel

```
OPTIONS MCG=(OVFLW,INDX,ZD=OFF)
```

oder

```
OPTIONS MCG=OVFLW,INDX,ZD=OFF
```

Die Optionszeichenkette des Parameters `MCG` kann mit einem Pluszeichen (+) oder einem Minuszeichen (-) beginnen, was bedeutet, dass die Werte von nicht erwähnten Optionen unverändert gelassen werden sollen und dass nur die vorhandenen Optionen gesetzt (+) oder zurückgesetzt (-) werden sollen, z.B.:

Beispiel:

```
  OPTIONS MCG=+PGEN         /* turns tracing on

 (statements to be traced)

OPTIONS MCG=-PGEN         /* turns tracing off
```

Beginnt die Zeichenkette mit einem anderen Zeichen als „+“ oder „-“, werden alle Optionen zurückgesetzt, bevor die Zeichenkette geparst wird.

Anmerkung: Außerdem bietet das Natural-Statement `OPTIONS` andere Natural-Compiler-Parameter als `MCG`.

Eine Erklärung der verwendeten Optionseinstellungen finden Sie im Abschnitt *Optimizer-Optionen*.

8 Optimizer-Optionen

Wenn der Natural Optimizer aktiviert wurde, können Sie Prüfungen angeben, indem Sie die in diesem Abschnitt erklärten Optionen einstellen.

Die Optionen können nicht benutzt werden, um Statements anzugeben, die optimiert werden sollen.

Liste der Optionen

Die folgende Tabelle enthält eine Liste mit den Beschreibungen der Natural Optimizer Compiler-Optionen. Unterstrichene Werte sind Standardwerte (Default Values), d.h. dieser Wert wird angenommen, wenn die Option nicht vorhanden ist.

Eine Natural Optimizer Compiler-Option besteht aus einer Zeichenkette, die von Klammern oder einzelnen Ausrufungszeichen umschlossen ist (Ausnahme: im Natural-Statement `OPTIONS`), wobei die Optionen durch Kommas voneinander abgetrennt sind. Einige Optionen haben Werte, während die bloße Existenz einiger Optionen in der Optionszeichenfolge ausreicht, um die Umgebung zu ändern.

Es gelten folgende Regeln:

- Optionale Klauseln sind von rechteckigen Klammern `[ ]` umgeben.
- Wahlmöglichkeiten sind von geschweiften Klammern `{ }` umgeben.
- Wahlmöglichkeiten sind voneinander durch senkrechte Linien `|`abgetrennt.
- Es kann nur eine dieser Wahlmöglichkeiten angegeben werden.

  `ON` ist gleichbedeutend mit `Y` (Yes),

  `OFF` ist gleichbedeutend mit `N` (No).
- Optionen, die ohne die optionale Klausel `ON` oder `OFF` (falls zutreffend) oder deren Äquivalenzwerte angegeben werden, werden so interpretiert, als ob sie auf `ON` gesetzt sind. Beispiel: `OVFLW` ist identisch zu `OVFLW=ON`.
- Mit Ausnahme der Option `OFF` gelten alle beim Optimieren angegebenen Optionsschalter (als ob `ON` angegeben wäre) und der Standardwert. Beispiel: `INDEX` ist identisch zu `ON,INDEX`.

Option	Erklärung
`ABEND`	Zwingt den Natural Optimizer Compiler, Code zu generieren, der bewirkt, dass Natural sofort abnormal beendet wird, wenn der Natural Optimizer Compiler während des Kompilierens die Option `ABEND` antrifft. Die Option muss selbständig erscheinen, sonst wird sie ignoriert. Andere Parameter werden durch diese Option nicht geändert oder zurückgesetzt. Diese Option kann für Debugging-Zwecke nützlich sein.

Option	Erklärung
ARCH	Gibt an, welcher Architecture Level für die Codegenerierung benutzt werden soll, siehe ***ARCH-Option*** im folgenden Abschnitt.
ARROPT	Gibt an, welche Generierung für Array-Zuweisungen des Typs A(*):=*scalar* verwendet werden soll, siehe ***ARROPT-Option*** im folgenden Abschnitt.
CACHE[={ON| OFF |Y|N}]	Schaltet die Variablenzwischenspeicherung (Variable Caching) ein oder aus. Siehe auch ***Variablenzwischenspeicherung (Caching)*** im Abschnitt *Performance-Überlegungen*.
CPU= /370	Gibt die Ziel-Architektur an.
DIGTCHCK[={ON| OFF |Y|N}]	Gibt an, ob die Ziffern von gepackten oder ungepackten Feldern (Formate P und N) geprüft werden sollen, wenn sie in eine andere Variable gleichen Typs und gleicher Genauigkeit verschoben werden. Wenn beispielsweise DIGTCHCK auf ON gesetzt ist und eine ungepackte Variable (Format N) eine ungültige Ziffer enthält, z.B. X'FA', dann generiert das Verschieben in eine andere ungepackte numerische Variable gleichen Typs den Fehler SOC7 (oder NAT0954). Wenn DIGTCHCK auf OFF gesetzt ist, wird kein Fehler generiert, aber der generierte Code ist viel schneller.
ERRDUMP[={ON| OFF |Y|N}]	Gibt an, ob der Natural Optimizer Compiler abbrechen soll, wenn während der Kompilierungsphase ein Fehler festgestellt wird. Dies ist nützlich, um am Natural Optimizer Compiler selbst eine Fehlerbehebung (Debugging) durchzuführen.
INDEX[={ON| OFF |Y|N}]	Gibt an, ob Array-Indizes auf Out-of-Bound-Werte im optimierten Code überprüft werden. Siehe auch die folgende ***Anmerkung***.
INDX[={ON| OFF |Y|N}]	Gibt an, ob Array-Indizes auf Out-of-Bound-Werte im optimierten Code überprüft werden. Zusätzlich wird RANGE auf ON gesetzt. Deshalb ist die Option gleichbedeutend mit INDEX=ON,RANGE=ON. Siehe auch die folgende ***Anmerkung***.
IO[={ON| OFF|Y|N}]	Nur aus Kompatibilitätsgründen vorhanden. Keine Wirkung.
LOOPS[={ON| OFF |Y|N}]	Nur aus Kompatibilitätsgründen vorhanden. Keine Wirkung.
MIX[={ON| OFF |Y|N}]	Nur aus Kompatibilitätsgründen vorhanden. Keine Wirkung.
NODBG[={ON| OFF|Y|N}]	Wenn NODBG=OFF/N (Standardeinstellung) gesetzt ist, kann der Natural Debugger benutzt werden, um Fehler im optimierten Code zu beheben (dann wird zusätzlicher Code generiert, um zu prüfen, ob der Test-Modus eingeschaltet worden ist (TEST ON). Bei NODBG=ON/Y wird weniger Code generiert, das Programm läuft schneller und verbraucht weniger CPU-Zeit. Anderseits wird die Funktionalität des Natural Debugger eingeschränkt, weil der Natural Debugger möglicherweise keine Steuerungsmöglichkeit bei optimierten Statements erhält.

Option	Erklärung
	Siehe auch NODBG im Abschnitt *Performance-Überlegungen*.
NOSGNTR[={ON| OFF |Y|N}]	Gilt nur bei gepackten Zahlen. Bei NOSGNTR=OFF (Standardeinstellung) werden Vorzeichen von gepackten Zahlen, die das Ergebnis einer arithmetischen Operation oder das Ziel einer Zuweisung sind, gemäß dem Subparameter PSIGNF (Interne Darstellung des positiven Vorzeichens bei gepackten Zahlen) des Systemkommandos COMPOPT gesetzt. Bei NOSGNTR=ON werden die aus der Ausführung der generierten Maschineninstruktion resultierenden Vorzeichen unverändert gelassen. Siehe auch ***Einfluss anderer Natural-Parameter***.
ON	Schaltet die Optimierung ein. Wenn keine zusätzliche Option angegeben wird, gilt der für jede Option definierte Standardwert. Wie in der folgenden Anmerkung mitgeteilt, kann dies unerwünschte Ergebnisse zur Folge haben. Dies gilt insbesondere für die Optionen INDEX, INDX, OVFLW und RANGE.
OFF	Schaltet die Optimierung aus.
OPTLEV={ 2|3}	Gibt die Optimierungsstufe an - entspricht in etwa der Anzahl der Durchläufe durch das Programm. OPTLEV=3 ist nützlich, wenn PGEN angegeben ist, da einige Verzweigungsziele beim ersten Durchlauf nicht ermittelt werden können und die PGEN-Ausgabe während des letzten Durchgangs erfolgt. Daher können einige Werte möglicherweise nicht korrekt angezeigt werden.
OVFLW[={ON| OFF |Y|N}]	Gibt an, ob Überlaufprüfungen bei aritmetischen Operationen oder Zuweisungen in den optimierten Code aufgenommen werden. Siehe auch die folgende ***Anmerkung***.
PGEN[={ON| OFF |Y|N}]	Gibt an, ob eine Disassemblierung des optimierten Code ausgegeben werden soll. Mit dieser Option werden auch alle anderen Nachverfolgungsoptionen aktiviert, siehe ***PGEN Option*** im folgenden Abschnitt.
RANGE[={ON| OFF |Y|N}]	Gibt an, ob bei Operationen mit Arrays Bereichsprüfungen durchgeführt werden. Dadurch wird sichergestellt, dass Array-Bereiche eine gleiche Anzahl an Elementen in allen entsprechenden Dimensionen aller Operanden haben. Siehe auch die folgende ***Anmerkung***.
SIGNCHCK[={ ON|OFF|Y|N}]	Gibt an, ob das Ergebnis einer Multiplikation mit einem gepackten oder nicht gepackten Multiplikator auf eine negative Null geprüft werden soll. Wenn Null mit einer negativen Zahl multipliziert wird, generiert die Maschineninstruktion MP ein negatives Null-Ergebnis. Wenn SIGNCHCK auf ON gesetzt ist, wird diese negative Null in eine positive Null umgewandelt. Die Prüfung auf eine negative Null erfolgt bei jeder Multiplikation mit einem gepackten oder nicht gepackten Multiplikator.
TRENTRY	Nur für den internen Gebrauch durch Software AG bestimmt. Die Einstellung dieses Parameters nicht ändern.

Option	Erklärung
UNICC	Gibt an, ob optimierter Code für IF- und DECIDE-Statements mit Unicode-Operanden generiert wird, siehe ***UNICC Option*** im folgenden Abschnitt.
ZD[={ ON|OFF|Y|N}]	Gibt an, ob Divisoren auf Null geprüft werden sollen. Ist diese Option angegeben, dann wird Code eingefügt, damit sich das Programm gemäß dem Natural-Profilparameter ZD verhält, d.h., es wird der Natural-Fehler NAT1302 ausgegeben oder das Ergebnis ist Null. Ist diese Option nicht angegeben, tritt der Natural-Fehler NAT0954 auf, wenn der Divisor Null ist. Siehe auch *ZD - Division durch Null* in der Natural *Parameter-Referenz*-Dokumentation.

Anmerkung zu INDEX, INDX, OVFLW und RANGE

Wenn die Option INDEX, INDX, OVFLW oder RANGE gesetzt ist, werden zusätzliche Instruktionen zu dem generierten Code hinzugefügt, um Datenüberlauf- und Index-außerhalb-Bereich-Situationen zu entdecken, falls solche während der Programmausführung auftreten sollten. Zwar vergrößert die Benutzung dieser Optionen den generierten Code, sie wird aber empfohlen, um sicherzustellen, dass fehlerhafte Programme aufgespürt werden und nicht zu unvorhersehbaren Ergebnissen, Speicherbeschädigungen oder abnormalen Programmabbrüchen führen können.

- Beispiel für INDEX und OVFLW
- Optimale Code-Generierung

Beispiel für INDEX und OVFLW

```
DEFINE DATA LOCAL
...
1 P1 (P1/9)
...
1 P3 (P3/9)
...
1 I (I4)
1 J (I4)
1 K (I4)
1 L (I4)
END-DEFINE
...
P1(I:J) := P3(K:L)
...
END
```

Erklärung des Beispiels:

Bei INDX=ON oder INDEX=ON wird Code generiert, um zu überprüfen, dass I, J, K und L innerhalb der für P1 bzw. P3 definierten Bereiche sind.

Bei INDX=ON oder RANGE=ON wird Code generiert, um zu überprüfen, dass I:J und K:L Bereiche von der gleichen Länge bezeichnen.

Bei OVFLW=ON wird Code generiert, um zu überprüfen, dass der Wert von P3 in die entsprechende P1-Variable passt.

Zum Beispiel: Der Wert 100 würde hier einen Überlauf verursachen.

Beispiel-Fehlersituation:

Wenn bei gesetztem OVFLW=OFF eine der Ausprägungen von P3 den Wert 100 enthält, ist der Wert, der der entsprechenden P1-Ausprägung zugewiesen wird, Null. Ist bei gesetztem INDX=OFF die Index-Variable I Null oder größer als 9, werden Speicherbereiche, die nicht zu Array P1 gehören, beschädigt. Sind diese Optionen (OVFLW und INDX) auf ON gesetzt, tritt ein Natural-Fehler auf, so wie es in der Standard-Natural-Laufzeit der Fall ist.

Für die oben angegebenen NOC-Optionen wird zusätzlicher Code generiert. Dies wird jedoch durch den Vorteil einer Prüfung gut kompensiert, die z. B. vor schwer zu entdeckenden und bereinigenden Fehlern schützt. Unentdeckte Fehler können natürlich zu unvorhersehbaren Ergebnissen führen.

Optimale Code-Generierung

Um sicherzustellen, dass die geringste Menge an Code generiert wird, und somit optimale Leistung zu erreichen, verwenden Sie:

```
OPT='NODBG,NOSGNTR,SIGNCHCK=OFF,ZD=OFF'
```

Wenden Sie diese Einstellung jedoch nur auf Objekte an, die gründlich mit dem Debugger fehlerbereinigt wurden, siehe auch *Anmerkung zu INDEX, INDX, OVFLW und RANGE*.

ARCH-Option

Die ARCH-Option dient zur Angabe des IBM Hardware Architecture Level, der zum Generieren von Code für ausführbare Natural-Objekte benutzt werden soll.

Wenn Sie einen ARCH-Wert angeben, generiert der Natural Optimizer Compiler neuere und schnellere Maschineninstruktionen, die die Leistung des generierten Codes verbessern können. Der angegebene Wert darf nicht höher sein als der Architecture Level Ihrer aktuellen Maschine. Ein mit einem ARCH-Level katalogisiertes ausführbares Natural-Objekt kann nur auf einer

Maschine mit gleichem oder einem höheren Architecture Level laufen. Deshalb empfehlen wir, die ARCH-Option nicht zu verwenden, wenn die katalogisierten Objekte auf einer beliebigen Maschine, insbesondere auf einer Maschine mit niedrigerem Architecture Level, ausgeführt werden sollen.

Ausführliche Informationen zu IBM Hardware Architecture Level siehe IBM-Literatur (*z/Architecture, Principles of Operation*).

Die folgenden Architecture Levels werden von der ARCH-Option des Natural Optimizer Compiler unterstützt:

IBM Architecture Level	Erforderliche IBM Hardware Facility
0	Gibt an, dass kein Architecture Level benutzt wird. Dies ist die Standardeinstellung für alle von Natural unterstützen Großrechnerplattformen.
1 bis 4	Diese Werte werden nicht ausgewertet und wie ARCH=0 behandelt.
5 bis 6	■ z800 oder z900 Extended-Translation Facility 2 ■ z890 oder z990 HFP Multiply-and-Add/Subtract Facility
7	■ z9 bis z109 Extended-Immediate Facility
8	■ z10 General-Instructions-Extension Facility ■ z10 Execute-Extensions Facility
9	■ zEnterprise 196 Load/Store-on Condition Facility Floating-Point-Extension-Facility Distinct-Operands Facility High-Word-Facility
10	■ zEnterprise EC12 (zEC12) Decimal Floating-Point Facility Decimal Floating-Point Zoned-Conversion Facility
11	■ zEnterprise z13 Decimal Floating-Point Packed-Conversion Facility
12	■ zEnterprise z14 Vector Packed-Decimal Facility

Anmerkung: Bei einem ARCH-Wert größer als Null generiert der Natural Optimizer Compiler Instruktionen bis hin zu dem in obiger Tabelle beschriebenen Facility Level. Ein ARCH-Wert größer als der Architecture Level der zugrunde liegenden Maschine wird bei der Kompilierung abgelehnt. Der Versuch, ein Programm, das mit einem ARCH-Level kompiliert wurde, auf einer Maschine mit einem niedrigeren Architecture Level zu starten, führt zu einem Laufzeitfehler NAT1394. Informationen zu der aktuellen Maschine können Sie sich mit dem Systemkommando TECH anzeigen lassen.

Folgende Themen werden behandelt:

Unterstützung für Architecture Level 10

Bei gesetztem ARCH=10 generiert der Natural Optimizer Compiler Instruktionen, die von der Decimal-Floating-Point (DFP) Zoned-Conversion Facility für die in den folgenden Abschnitten beschriebenen numerischen Operationen zur Verfügung gestellt werden. Dies kann die Ausführungsgeschwindigkeit für Statements, die diese Operationen verwenden, erheblich verbessern.

Operationen optimiert durch ARCH=10

Die folgenden arithmetischen Operationen auf Variablen der Natural-Datenformate I (ganzzahlig), N (numerisch ungepackt) und P (gepackt numerisch) profitieren von ARCH=10.

- Wertzuweisungen:

  P:=I

  P:=N

  N:=I

  N:=N

  N:=P nur wenn die Anzahl gepackter Ziffern kleiner gleich 15 ist.

  I:=N

- Arithmetische Operationen wie zum Beispiel mit den Statements ADD, SUBTRACT, DIVIDE und MULTIPLY, jedoch nur wenn folgende Bedingungen zutreffen:

  Mindestens einer der verwendeten Operanden ist im Format N oder I.
  Das Ergebnis der Operation überschreitet nicht 34 Stellen (Ganzzahl- + Dezimalstellen).

- Vergleiche wie zum Beispiel mit den Statements IF und DECIDE, jedoch nur wenn folgende Bedingungen zutreffen:

Mindestens einer der verwendeten Operanden ist im Format N.
Die beiden Operanden habe verschiedene Formate.

Unterstützung für Architecture Level 11

Bei gesetztem `ARCH=11` verwendet der Natural Optimizer Compiler Maschineninstruktionen, die mit der DFP Packed-Conversion Facility eingeführt wurden. Zusätzlich zu den mit `ARCH=10` optimierten numerischen Operationen optimiert `ARCH=11` auch Operationen, die nur gepackte Variablen verwenden.

Unterstützung für Architecture Level 12

Bei gesetztem `ARCH=12` generiert der Natural Optimizer Compiler Maschineninstruktionen, die mit der Vector Packed-Decimal Facility (VPD) eingeführt wurden. Dies kann die Ausführungsgeschwindigkeit bei Zuweisungen, Vergleichen und Berechnungen verbessern, wenn mindestens ein gepackter Operand verwendet wird.

VPD-Maschineninstruktionen werden bei den gleichen Natural-Operationen generiert, die bei **Architecture Level 10** und **Architecture Level 11** benutzt werden, sie werden jedoch nur bei arithmetischen Operationen verwendet, deren Ergebnisse 31 Stellen (Ganzzahlstellen + Dezimalstellen) nicht übersteigen.

Kompatibilität für Architecture Level 10 und 11

Wenn `ARCH=10` benutzt wird, generiert der Natural Optimizer Compiler Maschineninstruktionen, die mit der Decimal-Floating-Point (DFP) Zoned-Conversion Facility oder der DFP Packed-Conversion Facility eingeführt wurden. Diese Instruktionen bewirken eine schnellere Ausführung als die Standard-Maschinencode-Instruktionen für arithmetische Operationen, sie akzeptieren jedoch keine Daten, die in Bezug auf den gezonten numerischen Datentyp (N) unzulässig sind.

Dies kann zu Laufzeitfehlern führen, wenn ein N-Feld innerhalb eines `REDEFINE`-Abschnitts einer alphanumerischen oder binären Variablen definiert ist und das N-Feld nicht zulässig initialisiert wird, bevor es in einer arithmetischen Operation benutzt wird.

Ein numerisches zoniertes Feld enthält eine Ziffer in einem Byte. Normalerweise enthält jedes Byte `x'F'` im linken Halbbyte (**Z**one Bits) und den Ziffernwert (0-9) im rechten Halbbyte (**N**umeric Bits). Dies gilt für alle Bytes, außer dem letzten, das (A-F) im linken Halbbyte enthält (**S**ign Bits).

Ein Vorzeichen-Halbbyte (C,A,F,E) stellt einen positiven Wert dar, während (B,D) für einen negativen Wert steht. Ein anderer Wert als (0-9) innerhalb des numerischen Halbbytes (N) und ein anderer Wert als (A-F) innerhalb des Vorzeichen-Halbbytes (S) wird als ungültig angesehen. Die Daten innerhalb der Zonen-Halbbytes (Z) werden von arithmetischen Konvertierungsanweisungen nicht berücksichtigt und können einen beliebigen Wert (0-F) haben.

Beispiel für eine Variable, die als (N6) definiert ist:

ZN	ZN	ZN	ZN	ZN	SN	Vorzeichen ist	Wert ist	Funktioniert bei ARCH<=9	Funktioniert bei ARCH>=10
F1	F2	F3	F4	F5	F6	F=positive	123456, ok	ja	ja
F3	F2	F6	F3	F3	D2	D=negative	323662, ok	ja	ja
40	40	40	40	40	40	4=ungültig	000000, ok	ja	NAT7024
00	00	00	00	00	00	0=ungültig	000000, ok	ja	NAT7024
12	13	14	15	16	17	1=ungültig	234567, ok	ja	NAT7024
51	6B	72	7A	12	F1	F=positive	1B2A21, ungültig	NAT0954	NAT7024

Wenn ARCH=9 (oder niedriger) benutzt wird, werden ungültige Vorzeichen-Halbbytes (0-9) durch den generierten Code automatisch in ein positives Vorzeichen (F) korrigiert. Dadurch werden N-Felder mit leerem Inhalt zu gültigen Daten mit dem Wert Null. Das Gleiche gilt für Hexa-Null-Daten.

Wenn ARCH=10 oder ARCH=11 benutzt wird, bleiben ungültige Vorzeichen-Halbbytes (0-9) unverändert und führen zu einer Programmprüfung (Data Exception), wenn auf sie von einer DFP-Instruktion zugegriffen wird. Wenn ein solcher Abbruchfehler auftritt, gibt Natural anstelle des Fehlers NAT0954 den Fehler NAT7024 aus, um eindeutig anzuzeigen, dass der Fehler durch eine N-Variable verursacht wird, die keine gültigen numerischen Daten enthält.

Wenn ein numerisches Halbbyte (N) nicht den Wert (0-9) enthält, erfolgt unabhängig vom benutzten Architecture Level eine Programmprüfung (Data Exception).

Schlussfolgerung:

Benutzen Sie ARCH=10 oder ARCH=11 nicht mit einem anderen Vorzeichenwert als (A-F), um ein Programm zu katalogisieren, das mit unsauberen numerischen Daten arbeitet.

Beispiel:

```
OPTIONS MCG=(PGEN,ARCH=9)
DEFINE DATA LOCAL
1 #A (A6)
1 REDEFINE #A
  2 #N (N6)
END-DEFINE                                        /*  ARCH=9  ARCH=10
#A := H'F1F2F3F4F5F6'   ADD 1 TO #N  WRITE #N  /*  ok       ok
#A := H'F3F2F6F3F3D2'   ADD 1 TO #N  WRITE #N  /*  ok       ok
#A := H'404040404040'   ADD 1 TO #N  WRITE #N  /*  ok       NAT7024
#A := H'000000000000'   ADD 1 TO #N  WRITE #N  /*  ok       NAT7024
#A := H'121314151617'   ADD 1 TO #N  WRITE #N  /*  ok       NAT7024
#A := H'516B727A12F1'   ADD 1 TO #N  WRITE #N  /*  NAT0954  NAT7024
END
```

Darüber hinaus, wenn ARCH=10 (oder höher) benutzt wird, kann Natural einen Fehler NAT1305 (abgeschnittener numerischer Wert) anstelle eines Fehlers NAT1301 (Zwischenergebnis zu groß) aus folgendem Grund ausgeben: Zur Berechnung von Zwischenergebnissen wird das numerische

DFP-Format benutzt und ein Überlauf wird erst am Ende der arithmetischen Operation festgestellt, wenn das DFP in das Format des Ereignisses umgewandelt wird.

Kompatibilität für Architecture Level 12

Wenn `ARCH=12` benutzt wird, generiert der Natural Optimizer Compiler Maschineninstruktionen, die mit der Vector Packed-Decimal Facility (VPD) eingeführt wurden und die in Bezug auf die Unrichtigkeit der Daten mit dem Code kompatibel sind, der mit `ARCH=9` (oder niedriger) generiert wurde.

Numerische Datenfelder (N) mit unrichtigen Vorzeichendarstellungen (0-9) werden in den positiven Vorzeichenwert (F) konvertiert. Dabei werden numerische Felder mit Leerzeichen oder einem `hex00`-Inhalt akzeptiert und als Wert Null behandelt. Eine Daten-Ausnahme (Abend) tritt in dieses Fällen nicht auf.

ARROPT-Option

Die Option `ARROPT` bestimmt den Generierungsalgorithmus, der bei Array-Zuweisungen des Typs `A(*):=scalar` verwendet werden soll.

Gültige Werte für `ARROPT`:

Wert	Erklärung
`ON`	Empfohlene Einstellung für Arrays mit mindestens 50 Ausprägungen. `ON` ist die Standardeinstellung.
`OFF`	Empfohlene Einstellung für Arrays mit weniger als 50 Ausprägungen.

PGEN-Option

Die `PGEN`-Option bewirkt, dass der Natural Optimizer Compiler den generierten Code und interne Natural-Strukturen ausgibt. Dies ermöglicht es, Code und Strukturen zu untersuchen, z.B. zur Fehlerbehebung, Performance-Überprüfung und bei Support-Fragen.

Um die von der `PGEN`-Option gelieferten Ergebnisse interpretieren zu können, ist ein Verständnis der Arbeitsweise des /370-Assemblers von IBM erforderlich.

Wir empfehlen Ihnen, diese Option mit Unterstützung durch Ihren lokalen Software AG-Vertreter zu benutzen.

- PGEN setzen
- Sub-Optionen der PGEN-Option

PGEN setzen

Um die PGEN-Funktion zu benutzen, müssen Sie die PGEN-Option setzen, wenn Sie den Optimizer Compiler aktivieren.

Da der Pufferbereich im Speicher vorgehalten wird, kann es vorkommen, dass der Benutzer-Thread nicht groß genug ist, um die Trace-Informationen aufzunehmen. In diesem Fall sollten Sie versuchen, PGEN nur für den Teil des Programms zu setzen, dessen Ablauf verfolgt werden soll, zum Beispiel:

`OPTIONS MCG=(PGEN=ON,TRGPT=ON)`
oder
`OPTIONS MCG=+PGEN,TRGPT`
	Schaltet die Ablaufverfolgung (Tracing) ein, einschließlich der Ablaufverfolgung der GPT-Einträge.

`OPTIONS MCG=(PGEN=OFF)`
oder
`OPTIONS MCG=-PGEN`
	Schaltet die Ablaufverfolgung aus.

Es sind verschiedene Optionen verfügbar, die den Inhalt der Ausgabe beeinflussen. In ihrer Grundform bewirkt die PGEN-Option eine formatierte Auflistung von Natural-Sourcecode-Zeilen und eine Disassemblierung des entsprechenden Codes, der generiert und im Speicher abgelegt werden soll, um von dort mittels der NOCSHOW-Utility extrahiert zu werden, siehe Beschreibung in *Ausgabe der PGEN-Option*.

Die Optionen TRSTMT, TRGPT, TRMPT und TRVDT bewirken, dass Hex-Dumps von internen Datenstrukturen, die mit jeder Zeile verbunden sind, ausgegeben werden.

Die Optionen TRBASES und TRCACHE bewirken, dass Informationen über Basisregister und Cache-Variablen ausgedruckt werden.

Sub-Optionen der PGEN-Option

In der folgenden Tabelle werden die Optionen beschrieben, die bei PGEN=ON verfügbar sind. Die Erklärung der verwendeten Syntax finden Sie in der Einleitung zum Abschnitt *Liste der Optionen* weiter oben.

Option	Erklärung
`LPP={5|..| 55 |..|255}`	Zeilen pro Seite für die Trace-Ausgabe. Nur bei TREXT=ON benutzt.
`NOsrcE[={ON| OFF |Y|N}]`	Bei NOsrcE=OFF wird das Natural-Quellcode-Statement in die Ausgabe aufgenommen.
`TRACELEV={ 0 |..|255}`	Gibt den Trace Level an. Jedes Bit in diesem Ein-Byte-Wert gibt einen Puffertyp für die Verfolgung an; diese Bits können mit den Optionen TR*xxx* ebenfalls eingeschaltet werden.

Option	Erklärung
TRBASES[={ON| OFF |Y|N}]	Gibt an, ob Basisregisterzuweisungen verfolgt werden.
TRCACHE[={ON| OFF|Y|N}]	Gibt an, ob CACHE-Einträge verfolgt werden.
TREXT[={ON| OFF |Y|N}]	Bei TREXT=ON wird die Ablaufverfolgung an den User Exit NOCPRINT geleitet, Beschreibung siehe unten.
TRGPT[={ON| OFF |Y|N}]	Gibt an, ob GPT-Einträge verfolgt werden.
TRMPT[=ON| OFF |Y|N}]	Gibt an, ob MPT-Einträge verfolgt werden.
TRSTMT[={ON| OFF |Y|N}]	Gibt an, ob STMT-Einträge verfolgt werden.
TRVDT[={ON| OFF |Y|N}]	Gibt an, ob VDT-Einträge verfolgt werden.

Siehe auch die folgenden Beispiele.

Ausgabe der PGEN-Option

Es gibt zwei Stellen, an die der Natural Optimizer Compiler die Ausgabe von PGEN leiten kann.

- Interner Pufferspeicher
- User Exit NOCPRINT

Interner Pufferspeicher

Der Inhalt dieses Pufferspeichers wird jedes Mal überschrieben, wenn eines der Kommandos CHECK, CAT, STOW oder RUN ausgeführt wird.

Es wird ein Systemdienstprogramm (Utility) NOCSHOW bereitgestellt, mit dem der Inhalt dieses Puffers angesehen, durchsucht oder gedruckt werden kann.

Um die Utility NOCSHOW aufzurufen:

- Geben Sie das Direktkommando NOCSHOW ein, nachdem eines der Kommandos CHECK, STOW, CAT oder RUN ausgeführt worden ist, während der Natural Optimizer Compiler aktiv gewesen ist.

Im Bildschirm sind folgenden PF-Tasten belegt:

PF-Taste	Funktion
PF2	Position am oberen Rand der Ausgabe.
PF4	Position eine Zeile zurück
PF5	Position eine Zeile vor
PF6	In Report (1) drucken
PF7	Position eine Seite zurück
PF8	Position eine Seite vorwärts

PF-Taste	Funktion
PF9	Via Entire Connection in Report (7) drucken
PF10	Nach Zeichenkette durchsuchen
PF11	Suche wiederholen

User Exit NOCPRINT

Wenn `TREXT=ON` angegeben ist, übergibt der Natural Optimizer Compiler jede Ausgabezeile an den User Exit `NOCPRINT`, anstatt sie im Trace-Pufferspeicher hinzuzufügen.

Der Aufruf von `NOCPRINT` erfolgt gemäß normalen Betriebssystem-Registerkonventionen. Register 1 zeigt auf ein Vollwort, das die Adresse der 81-Byte-Druckzeile mit ANSI-Vorschubsteuerzeichen in Position 1 enthält. Register 13 zeigt auf einen Bereich von 18*4 Bytes, der als Speicherregister benutzt werden kann. Register 14 enthält die Rückgabeadresse und Register 15 enthält die Eingabeadresse von `NOCPRINT`.

Der User Exit `NOCPRINT` kann in einer beliebigen Sprache geschrieben sein, die die oben genannten Registerkonventionen unterstützt. Der User Exit muß zusammen mit dem Natural Optimizer Compiler-Nukleus mit dem Natural-Nukleus verlinkt sein.

Arbeiten mit der PGEN-Ausgabe

Dieser Abschnitt enthält Hinweise und Erläuterungen, wie die mit der `PGEN`-Option erstellte Ausgabe zu interpretieren ist.

- Oben in der `PGEN`-Ausgabe stehen einige disassemblierte Zeilen, die scheinbar nicht zu einer Sourcecode-Zeile gehören. Es handelt sich um die Instruktionen, aus denen der Prolog besteht, der immer dann ausgeführt wird, wenn die Steuerung vom nicht-optimierten zum optimierten Code übergeben wird. Permanente Basisregister werden geladen und die Steuerung wird an den korrekten Punkt im Prolog übergeben. Siehe ***Beispiel Abschnitt A*** weiter unten.
- Manchmal werden viele Zeilen ohne Code gedruckt. Dies bedeutet, dass dort kein Code erforderlich war oder das diese Statements von der NOC-Optimierung ausgeschlossen sind. Siehe ***Beispiel Abschnitt B*** weiter unten.

  Besteht daraüber hinaus der für ein Natural-Statement generierte Code nur aus Folgendem:

  ```
  BAS   R14,RETH
  DC    X'....'
  ```

  so bedeutes dies eine Rückkehr zur Standardlaufzeit, weil dieses Statement nicht von NOC optimiert werden konnte (siehe Zeile `0170`).
- Wenn `NODBG=OFF` (Standardeinstellung) angegeben worden ist, wird zu Beginn eines jeden neuen Natural-Statement eine Instruktionsfolge generiert.

```
BALR R9,R11
DC X'....'
```

Diese Instruktionsfolge setzt (im Fehlerfall) die Zeilennummer und prüft, ob der Test-Modus eingeschaltet ist (`TEST` auf `ON`). Ohne diese Instruktionsfolge ist ein Debugging von NOC-kompilierten Statements durch den Natural Debugger nicht möglich. Siehe *Beispiel Abschnitt C* weiter unten.

- Manchmal ist zwischen disassemblierten Zeilen ein Zeilenumbruch vorhanden. Der Umbruch zeigt eine interne Statement-Trennung an. Dies kommt vor, weil ein einzelnes Natural-Statement oft mehrere interne (Pseudo-Code-)Statements generiert.
- Die bedienten Natural-Variablen werden in den Assembler-Code eingefügt.
- Die Angaben auf der rechten Seite (z.B. `START 8FEC`) sind interner Art. Sie dokumentieren den Pfad, wie der Code von den NOC-Modulen generiert wurde.
- Alle Adressarten innerhalb des Codes werden aufgelöst und in der Form `=(00044)` zur Verfügung gestellt. Dies dokumentiert den Versatz im Code, an dem die Verzweigung ausgeführt wird.
- Die erste und die letzte Code-Instruktion enthält die zum Kompilieren dieses Programms benutzte NOC-Version: `4700 8410` bedeutet NOC-Version V841.

Beispiel Abschnitt A:

```
000000 4700 8410           NOP   1040(,R8)                        START  8FEC
000004 5880 D354           L     R8,CONST                               D9DC
000008 5870 D370           L     R7,LOCAL                               D9DC
00000C 4810 6006           LH    R1,6(,R6)                              90A0
000010 1F60                SLR   R6,R0                                  90BA
000012 47F1 A000           B     0(R1,R10)                              90C0

000016 4DE0 B040           BAS   R14,RETH                       RETN   F0AA
00001A 0034                DC    X'0034'                                F0C0
```

Beispiel Abschnitt B:

```
0010 0010 OPTIONS MCG=(PGEN,OVFLW,INDX)
 0020 DEFINE DATA LOCAL
 0030 1 I(I4)
 0040 1 P(P7.2)
 0050 1 T(P7.2)
 0060 END-DEFINE
 0070 *
 0080 SETTIME
 0090 *      ↩
```

Beispiel Abschnitt C:

```
0100 FOR I=1 TO 100000                                                  ↩


00001C 0D9B               BASR  R9,R11                            MOVE 1724A
00001E 004A               DC    X'004A'                                17278
000020 D203 7000 8148     MVC   I(4),#KST0148                           97D2

000026 47F0 A044          B     68(,R10)          =(00044)          GOTO  EF44

00002A 0D9B               BASR  R9,R11                             ADD 1724A
00002C 006A               DC    X'006A'                                17278
00002E BF0F 7000          ICM   R0,B'1111',I                            BB20
000032 5A00 8148          A     R0,#KST0148                             1E4E
000036 0D90               BASR  R9,0                                    F12A
000038 4710 B15C          BO    NAT1301                                 1E9E
00003C BE0F 7000          STCM  R0,B'1111',I                            A9CC

000040 0D9B               BASR  R9,R11                               IF 1724A
000042 007C               DC    X'007C'                                17278
000044 BF0F 7000          ICM   R0,B'1111',I                            BB20
000048 5900 819B          C     R0,#KST019B                             3FDA
00004C 47D0 A054          BNH   84(,R10)          =(00054)            EF44

000050 47F0 A078          B     120(,R10)         =(00078)          GOTO  EF44   ↩


0110   ADD 1.00 TO P

000054 0D9B               BASR  R9,R11                             ADD 1724A
000056 0092               DC    X'0092'                                17278
000058 FA41 7004 819F     AP    P(5),#KST019F(2)                        20A0
00005E 0D90               BASR  R9,0                                    F12A
000060 4710 B15C          BO    NAT1301                                1071C
000064 910D 7008          TM    P+4,X'0D'                              120B0
000068 4710 A070          BO    112(,R10)          =(00070)            120F6
00006C 960F 7008          OI    P+4,X'0F'                              1210ª

0120 END-FOR
0130 *

000070 0D9B               BASR  R9,R11                            GOTO 1724A
000072 00A4               DC    X'00A4'                                17278
000074 47F0 A02A          B     42(,R10)          =(0002A)              EF44   ↩


0140 T:=*TIMD(0080)

000078 0D9B               BASR  R9,R11                            SYFU 1724A
00007A 00AE               DC    X'00AE'                                17278
```

```
00007C 4DE0 B0D8           BAS   R14,SYSFUNC                              5F1A
000080 0190 B881           DC    X'0190B881'                              5F28

000084 F246 7009 8190      PACK  T(5),#KST0190(7)                  MOVE  AD18
00008A 910F 700D           TM    T+4,X'0F'                               12130
00008E 4710 A0A0           BO    160(,R10)        =(000A0)               12176
000092 17EE                XR    R14,R14                                 1218E
000094 43E0 700D           IC    R14,T+4                                 12194
000098 43EE B488           IC    R14,PSGNTR(R14)                         121AA
00009C 42E0 700D           STC   R14,T+4                                 121B2
0000A0 F040 7009 0002      SRP   T(5),2,0                                 ACA2
0000A6 17EE                XR    R14,R14                                 1218E
0000A8 43E0 700D           IC    R14,T+4                                 12194
0000AC 43EE B488           IC    R14,PSGNTR(R14)                         121AA
0000B0 42E0 700D           STC   R14,T+4                                 121B2   ↩


0150 T:=T / 10
0160 *

0000B4 0D9B                BASR  R9,R11                             DIV 1724A
0000B6 00C0                DC    X'00C0'                                 17278
0000B8 F864 D100 7009      ZAP   OP1(7),T(5)                              AC60
0000BE FD61 D100 81A1      DP    OP1(7),#KST01A1(2)                       327A
0000C4 F844 7009 D100      ZAP   T(5),OP1(5)                              AC60
0000CA 910D 700D           TM    T+4,X'0D'                               120B0
0000CE 4710 A0D6           BO    214(,R10)        =(000D6)               120F6
0000D2 960F 700D           OI    T+4,X'0F'                               1210A   ↩


0170 DISPLAY 'ELAPSED TIME (S)' T

0000D6 4DE0 B040           BAS   R14,RETH                           RETN  F0AA
0000DA 00D2                DC    X'00D2'                                 F0C0

0180 END

0000DC 40D6 D7E3 F844 2000 DC    X'40D6D7E3F8442000'  =' OPT8à..'    END  927E
0000E4 0000 0000           DC    X'00000000'                               nf
0000E8 40D5 D6C3 F8F4 F140 DC    X'40D5D6C3F8F4F140'  =' NOC841 '         92E4
```

UNICC-Option

Die UNICC-Option steuert die Generierung von optimiertem Code für IF-, DECIDE FOR- und DECIDE ON-Statements, die Unicode-Operanden enthalten.

Gültige Werte für UNICC:

Wert	Erklärung
ON	Generiert optimierten Code und prüft, ob COLLATE=OFF gesetzt ist (siehe Profilparameter CFICU in der *Parameter-Referenz*-Dokumentation). Falls COLLATE=ON gesetzt ist, schlägt die Ausführung des optimierten Code fehl und der Natural-Systemfehler NAT7023 tritt auf.
FORCE	Generiert optimierten Code wie bei ON, aber ohne abzuprüfen, ob COLLATE=OFF gesetzt ist.. Der mit FORCE optimierte Code hat eine bessere Leistung als der mit ON optimierte, kann jedoch falsche Ergebnisse verursachen, wenn COLLATE=ON gesetzt ist.
OFF	Es wird kein optimierter Code generiert. OFF ist die Standardeinstellung.

Voraussetzungen für die Code-Generierung mit Unicode-Operanden

Der Natural Optimizer Compiler generiert optimierten Code für Natural-Statements mit Unicode-Zeichenketten, wenn die folgenden Erfordernisse erfüllt sind:

Statement	Erforderlich
Alle Statements	Alle im Statement benutzten Operanden müssen vom Typ Unicode sein.
EXAMINE	Die ARCH-Option muss auf einen Wert größer gleich 6 gesetzt sein.
IF DECIDE FOR DECIDE ON	■ Alle Unicode-Zeichenketten müssen normalisiert sein. ■ Die ARCH-Option muss auf einen Wert größer gleich 5 gesetzt sein. ■ Die **UNICC-Option** muss auf ON oder FORCE gesetzt sein. ■ Die COLLATE-Option des Profilparameters CFICU muss auf OFF gesetzt sein (siehe *Parameter-Referenz*-Dokumentation).
MOVE MOVE SUBSTRING RESET	Die ARCH-Option muss auf einen Wert größer gleich 5 gesetzt sein.

Einfluss anderer Natural-Parameter

Der globale Parameter ZD beeinflusst das Verhalten des NOC-Compiler. Siehe Beschreibung der ZD-Option unter ***Liste der Optionen*** weiter oben.

Der COMPOPT-Subparameter PSIGNF (siehe auch Systemkommando COMPOPT in der Natural-*Systemkommando*-Dokumentation) beeinflusst das Verhalten, indem er erzwingt, dass positive gepackte Dezimalzahlen bei ON auf F und bei OFF auf C gesetzt werden. Der Parameter wird angewendet, wenn NOSGNTR=OFF angegeben ist.

Siehe die Tabelle unten für gepackte Daten (Format P) :

NOSGNTR=OFF	und	PSIGNF=ON	Alle Vorzeichen werden auf F normalisiert (Standard).
NOSGNTR=OFF	und	PSIGNF=OFF	Alle Vorzeichen werden auf C normalisiert.
NOSGNTR=ON			Alle Vorzeichen werden so belassen, wie sie bei der letzten Operation erzeugt wurden.

Für numerische Daten (Format N) werden die Vorzeichen immer auf F normalisiert, unabhängig von den Einstellungen von NOSGNTR und PSIGNF.

9 Performance-Überlegungen

Formate

Die beste Leistung wird erreicht, wenn Sie die Datenformate gepackt numerisch (P) und ganzzahlig (I4) bei arithmetischen Operationen verwenden.

Vermeiden Sie die Konvertierung von Daten zwischen den Formaten gepackt numerisch (P), ungepackt numerisch (N), Ganzzahl (I) und Fließkomma (F), da dies selbst bei optimiertem Code Verarbeitungsaufwand verursacht.

Da es bei optimiertem Code keinen Interpretationsaufwand gibt, werden die Unterschiede zwischen den verschiedenen Datenformaten viel deutlicher: Bei optimiertem Code ist die Leistungsverbesserung, die durch die Verwendung des Formats P anstelle von zum Beispiel N erzielt wird, noch höher als bei normalem Code.

Beispiel:

```
A = A + 1
```

In der obigen numerischen Berechnung

- mit nicht-optimiertem Code wird das Format P um ca. 13 % schneller als Format N ausgeführt,
- mit optimiertem Code wird das Format P jedoch um ca. 56 % schneller als Format N ausgeführt.

Die Leistungssteigerung, die durch Benutzung des Natural Optimizer Compiler bei diesem einzelnen Statement erzielt wird, ist:

- mit nicht gepackten Operanden (N): 8-mal schneller,
- mit gepackten Operanden (P): 15-mal schneller.

Arrays

Array-Bereichsoperationen, wie zum Beispiel

```
MOVE A(*) TO B(*)
```

werden effizienter ausgeführt, als wenn die gleiche Funktion unter Verwendung einer `FOR`-Statement-Verarbeitungsschleife programmiert worden wäre. Dies ist auch für optimierten Code gültig.

Bei Benutzungen von Indizes sollte das Ganzzahlformat I4 verwendet werden, um eine optimale Leistung zu erzielen.

Alphanumerische Felder

Wenn Sie eine alphanumerische Konstante in eine alphanumerische Variable (Format A) verschieben oder eine alphanumerische Variable mit einer alphanumerischen Konstanten vergleichen, empfehlen wir Ihnen, die Länge der alphanumerischen Konstante an die Länge der Variablen anzupassen. Dadurch wird die Operation erheblich beschleunigt, zum Beispiel:

```
A(A5):='XYZAB'


...
IF A = 'ABC  ' THEN ...
```

ist schneller als

```
IF A = 'ABC' THEN ...
```

DECIDE ON

Wenn Sie das `DECIDE ON`-Statement mit einer Systemvariablen, einem Array oder einem *`operand1`*-Parameter benutzen, ist es effizienter, den Wert in eine im `LOCAL`-Speicherabschnitt definierte Skalarvariable mit gleichem Typ und gleicher Länge zu verschieben.

Numerische Werte

Versuchen Sie, wenn Sie numerische Konstanten in Zuweisungen oder arithmetischen Operationen verwenden, die Konstanten zu veranlassen, denselben Typ zu haben wie die Operation.

Daumenregeln

- Jede numerische Konstante mit oder ohne Dezimalstelle, aber ohne Exponenten, wird zu einer gepackten Zahl mit der minimalen Länge und Genauigkeit zur Darstellung des Wertes kompiliert, es sei denn, die Konstante ist ein Array-Index oder eine Teilstring-Startposition oder -Länge, in diesem Fall wird sie zu einer 4-Byte-Ganzzahl (I4). Diese Regel gilt unabhängig von den an der Operation beteiligten Variablentypen.
- Operationen, die Fließkomma enthalten, werden in Fließkomma ausgeführt. Fügen Sie bei numerischen Werten `E00` hinzu, um zu erzwingen, dass sie Fließkomma sind, zum Beispiel:

```
ADD 1E00 to F(F8)
```

- Operationen, die kein Fließkomma, sondern gepackt numerische, ungepackt numerische, Datums- oder Zeit-Variablen enthalten, werden in gepackten Dezimalzahlen ausgeführt. Für `ADD`, `SUBTRACT` und `IF` erzwingen Sie, dass numerische Konstanten die gleiche Anzahl an Dezimalstellen wie die Variable mit der höchsten Genauigkeit haben, indem Sie eine Dezimalstelle und nachgestellte Nullen hinzufügen, zum Beispiel:

```
ADD 1.00 TO P(P7.2)
```

Bei `MULTIPLY` und `DIVIDE` ist diese Technik nicht nötig.

Variable Positionierung

Um den Optimierungsprozess zu erleichtern, sollten Sie versuchen, alle skalaren Referenzen am Anfang des Datenbereichs und alle Array-Referenzen am Ende des Datenbereichs zu halten.

Variablenzwischenspeicherung (Caching)

Der Natural Optimizer Compiler enthält einen Algorithmus, der die die Leistung noch weiter steigert. Hinsichtlich Leistung ist das Statement in Abhängigkeit von den Typen der Operanden unterschiedlich. Das Statement wird langsamer ausgeführt, wenn es sich bei einem oder mehreren der Operanden um einen Parameter, ein Array oder ein Skalarfeld des Typs N (numerisch) oder Kombinationen aus diesen handelt. Der Natural Optimizer Compiler analysiert den Programmfluss und bestimmt, welche Variablen mit einer oder mehreren dieser Kennzeichen zweimal oder mehrmals gelesen werden, ohne das in sie geschrieben wird. Der Natural Optimizer Compiler verschiebt dann den Wert jeder einzelnen Variablen in einen temporären Zwischenspeicherbereich, wo auf ihn unter folgenden Bedingungen schnell zugegriffen werden kann:

- Auf die Variable wird oft zugegriffen, sie wird aber selten geändert *und*
- die Variable ist ein Array eines beliebigen Typs oder ein Skalarfeld des Typs N (numerisch).

Am besten geeignet für die Variablenzwischenspeicherung (Variable Caching) sind Programme mit langen Sequenzen, die wiederholt auf dieselbe Variable zugreifen, insbesondere wenn die Variable ein Array ist. Durch die Variablenzwischenspeicherung wird dann eine komplexe und wiederkehrende Adressenberechnung vermieden.

Beispiel für Variablenzwischenspeicherung (Caching)

Das folgende Beispiel demonstriert den Vorteil der Variablenzwischenspeicherung. Die Ausführung dieses Programms, das mit NODBG (siehe unten) und CACHE=ON katalogisiert wurde, beanspruchte in einem Test 47 % der Zeit, die bei NODBG and CACHE=OFF benötigt wird. Wird das Programm mit CACHE=ON katalogisiert, verringert sich der NOC-generierte Code von 856 Bytes auf 376 Bytes.

```
DEFINE DATA LOCAL
1 ARR(N2/10,10,10)
1 I(I4) INIT <5>
1 J(I4) INIT <6>
1 K(I4) INIT <7>
END-DEFINE
DECIDE ON EVERY ARR(I,J,K)
 VALUE 10 IGNORE
 VALUE 20 IGNORE
 VALUE 30 IGNORE
 VALUE 40 IGNORE
 VALUE 50 IGNORE
 VALUE 60 IGNORE
 VALUE 70 IGNORE
 VALUE 80 IGNORE
 VALUE 90 IGNORE
 NONE IGNORE
END-DECIDE
```

Vorsicht: Wenn der Inhalt einer im Cache zwischengespeicherten Variablen mit dem Kommando MODIFY VARIABLE des Natural Debugger geändert wird, wird nur der Inhalt der ursprünglichen Variablen geändert. Der zwischengespeicherte Wert (der möglicherweise noch in nachfolgenden Statements benutzt wird) bleibt unverändert. Daher sollte Variablen-Caching mit großer Sorgfalt eingesetzt werden, wenn der Natural Debugger verwendet wird. Siehe auch die Natural-*Debugger*-Dokumentation.

NODBG

Wenn ein Programm gründlich getestet und in Produktion genommen wurde, sollten Sie das Programm mit der Option NODBG, wie im Abschnitt ***Optimizer-Optionen*** beschrieben, katalogisieren. Ohne Debug-Code werden die optimierten Statements um 10% bis 30% schneller ausgeführt.

Bei Angabe der Option NODBG wird der zum Erleichtern des Debug-Vorgangs vorhandene Code entfernt, und zwar auch bei eingeschalteten Optionen INDX und OVFLW.

10 Zaps auflisten

Wenn Sie sich einen Überblick über die Zaps verschaffen wollen, die auf den Natural Optimizer Compiler an Ihrem Standort angewendet wurden, verwenden Sie das Systemkommando `DUMP`.

Um einen Überblick über die Zaps zu erhalten:

- Geben Sie folgendes Natural-Systemkommando ein:

  ```
  DUMP ZAPS NOC
  ```

  Es wird eine Liste der Zaps angezeigt, die angewendet wurden.

Falls keine Zaps auf den Natural Optimizer Compiler angewendet wurden, erhalten Sie eine entsprechende Meldung.