COMPOPT

```
COMPOPT [option=value ...]
```

Mit diesem Kommando können Sie verschiedene Kompilierungsoptionen setzen. Die Optionen werden wirksam, wenn ein Natural-Programmierobjekt kompiliert wird.

Die Standardwerte für die einzelnen Optionen werden mit den entsprechenden Profilparametern in der Natural-Parameterdatei gesetzt.

Folgende Themen werden behandelt:

- Syntax-Erklärung
- Compiler-Optionen
- Compiler-Parameter angeben
- COMPOPT in einer Remote Mainframe-Umgebung
- Compiler-Schlüsselwortparameter angeben (Remote Mainframe-Umgebung)
- Allgemeine Compiler-Optionen (Remote Mainframe-Umgebung)
- Kompilierungsoptionen für Versionskompatibilität (Remote Mainframe-Umgebung)

Syntax-Erklärung

COMPOPT	Wenn Sie nur das Kommando `COMPOPT` eingeben, wird ein Dialogfeld angezeigt, in dem Sie die unten beschriebenen Optionen ein- bzw. ausschalten können. Die dort vorhandenen Schlüsselwörter werden nachfolgend beschrieben. Siehe auch *Compiler Options* in der Dokumentation *Natural Studio benutzen*.
COMPOPT ***option=value***	Anstatt eine Option im Dialogfeld zu setzen, können Sie sie auch direkt mit dem `COMPOPT`-Systemkommando angeben. **Beispiel:** `COMPOPT DBSHORT=ON`

Compiler-Optionen

Folgende Compiler-Optionen stehen zur Verfügung. Informationen über Zweck und mögliche Einstellungen sind in den Beschreibungen der gleichnamigen Natural-Profilparameter enthalten.

KCHECK | PCHECK | DBSHORT | PSIGNF | TQMARK | THSEP | GFID | MASKCME

Compiler-Parameter angeben

Compiler-Parameter können Sie auf verschiedenen Ebenen angeben:

1. **Als Standardeinstellungen**

   Die Standardeinstellungen der einzelnen Compiler-Parameter werden mit dem **Compiler Options**-Dialog in der Configuration Utility vorgenommen. Diese Einstellungen werden in der Natural-Parameterdatei NATPARM gespeichert.

2. **Zu Beginn einer Natural-Session**

   Zu Beginn einer Natural-Session können Sie die Standardeinstellungen überschreiben, indem Sie die entsprechenden Profilparameter setzen.

3. **Während einer aktiven Natural-Session**

   Während einer aktiven Natural-Session haben Sie zwei Mölichkeiten, die Compiler-Parameterwerte mit dem Systemkommando COMPOPT zu ändern: Entweder direkt mittels Kommandozuordnung (COMPOPT *option=value*) oder indem Sie das Kommando COMPOPT ohne Optionen absetzen, woraufhin das Dialogfeld **Compiler Options** erscheint. Bei einem LOGON auf eine andere Library, werden die Standardeinstellungen (siehe Punkt 1) wieder hergestellt. Beispiel:

   ```
    OPTIONS KCHECK=ON
   DEFINE DATA LOCAL 1
   #A (25) INIT <'Hello World'>
   END-DEFINE
   WRITE #A
   END
   ```

4. **In einem Natural-Programmierobjekt**

   In einem Natural-Programmierobjekt (z.B. Programm, Subprogramm) können Sie Compiler-Parameter mit dem OPTIONS-Statement setzen. Beispiel:

   ```
    OPTIONS KCHECK=ON WRITE 'Hello World'
   END
   ```

   Die in einem OPTIONS-Statement angegebenen Compiler-Optionen betreffen nur die Kompilierung dieses Programmierobjekts, sie aktualisieren jedoch nicht die mit dem Systemkommando COMPOPT gesetzten Optionen.

COMPOPT in einer Remote Mainframe-Umgebung

Die folgenden Themen gelten nur, wenn das COMPOPT-Kommando in einer Remote Mainframe-Umgebung abgesetzt wird.

Compiler-Schlüsselwortparameter angeben (Remote Mainframe-Umgebung)

Compiler-Schlüsselwortparameter können Sie auf verschiedenen Ebenen angeben:

1. **Als Standardeinstellungen**

   Die Standardeinstellungen der einzelnen Compiler-Schlüsselwortparameter werden mit dem Parameter-Makro NTCMPO vorgenommen. Diese Einstellungen werden im Natural-Parametermodul NATPARM gespeichert.

2. **Zu Beginn einer Natural-Session**

   Zu Beginn einer Natural-Session können Sie die Compiler-Schlüsselwortparameter überschreiben, indem Sie die Werte im entsprechenden Profilparameter CMPO setzen.

3. **Während einer aktiven Natural-Session**

   Während einer aktiven Natural-Session haben Sie zwei Möglichkeiten, die Compiler-Parameterwerte mit dem Systemkommando COMPOPT zu ändern: Entweder direkt mittels Kommandozuordnung (COMPOPT *option=value*) oder indem Sie das Kommando COMPOPT ohne Optionen absetzen, woraufhin der Schirm **Compiler Options** erscheint.

   Bei einem LOGON auf eine andere Library, werden die mit dem Makro NTCMPO und/oder dem Profilparameter CMPO vorgenommenen Standardeinstellungen (siehe Punkt 1) wieder hergestellt. Beispiel:

   ```
   OPTIONS KCHECK=ON
   DEFINE DATA LOCAL
   1 #A (A25) INIT <’Hello World’>
   END-DEFINE
   WRITE #A
   END
   ```

4. **In einem Natural-Programmierobjekt**

   In einem Natural-Programmierobjekt (z.B. Programm, Subprogramm) können Sie Compiler-Parameter mit dem OPTIONS-Statement setzen. Beispiel:

   ```
   OPTIONS KCHECK=ON
   WRITE ’Hello World’
   END
   ```

   Die in einem OPTIONS-Statement angegebenen Compiler-Optionen betreffen nur die Kompilierung dieses Programmierobjekts, sie aktualisieren jedoch nicht die mit dem Systemkommando COMPOPT gesetzten Optionen.

Allgemeine Compiler-Optionen (Remote Mainframe-Umgebung)

Folgende Optionen stehen zur Verfügung:

- KCHECK - Keyword Checking
- PCHECK - Parameter Checking for CALLNAT Statements
- DBSHORT - Interpretation of Database Short Field Names
- PSIGNF - Internal Representation of Positive Sign of Packed Numbers
- TSENABL - Applicability of TS Profile Parameter
- GFID - Generation of Global Format IDs
- LOWSRCE - Allow Lower-Case Source
- TQMARK - Translate Quotation Mark
- THSEP - Dynamic Thousands Separator
- CPAGE - Code Page Support for Alphanumeric Constants
- DB2ARRY - Support DB2 Arrays in SQL SELECT and INSERT Statements
- CHKRULE - Validierung von INCDIR-Statements in Maps

Diese Optionen entsprechen den gleichnamigen Schlüsselwortparametern des Profilparameters `CMPO` bzw. des Parameter-Makros `NTCMPO`.

KCHECK - Keyword Checking

Prüfung auf Schlüsselwörter.

ON	Felddeklarationen in einem Programmierobjekt werden gegen einen Satz kritischer Natural-Schlüsselwörter geprüft. Falls ein Variablenname mit einem dieser Schlüsselwörter übereinstimmt, wird ein Syntaxfehler ausgegeben, wenn das Programmierobjekt geprüft oder katalogisiert wird.
OFF	Es erfolgt keine solche Prüfung. Dies ist der Standardwert.

Der Abschnitt *Prüfung der für Natural reservierten Schlüsselwörter durchführen* im *Leitfaden zur Programmierung* enthält eine Liste der Natural-Schlüsselwörter, die von der `KCHECK`-Option abgeprüft werden.

Der Abschnitt *Alphabetische Liste der für Natural reservierten Schlüsselwörter* im *Leitfaden zur Programmierung* enthält eine Übersicht über alle Natural-Schlüsselwörter und reservierten Wörter.

PCHECK - Parameter Checking for CALLNAT Statements

Prüfung der Parameter bei Objekt aufrufenden Statements.

ON	Der Compiler prüft Anzahl, Format, Länge und Array-Index-Grenzen der Parameter, die in einem Objekt aufrufenden Statement (zum Beispiel `CALLNAT`, `PERFORM`, `INPUT USING MAP`, `PROCESS PAGE USING`, Helproutine-Aufruf) angegeben sind. Darüber hinaus wird bei der Parameterprüfung das `OPTIONAL`-Feature des `DEFINE DATA PARAMETER`-Statements berücksichtigt. Die Parameterprüfung basiert auf dem Vergleich der Parameter des Objekt aufrufenden Statements mit den `DEFINE DATA PARAMETER`-Definitionen in dem aufzurufenden Subprogramm. Dazu ist folgendes erforderlich: ● Der Name des aufzurufenden Subprogramms muss als alphanumerische Konstante (nicht als alphanumerische Variable) definiert sein. ● Das aufzurufende Subprogramm muss als katalogisiertes Objekt zur Verfügung stehen. Andernfalls hat die Einstellung `PCHECK=ON` keine Auswirkung. **Probleme bei der Benutzung des CATALL-Kammandos mit PCHECK=ON** Wenn ein `CATALL`-Kommando zusammen mit der Einstellung `PCHECK=ON` benutzt wird, ist folgendes zu beachten: Wenn ein `CATALL`-Vorgang aufgerufen wird, hängt die Reihenfolge, in der die Programmierobjekte kompiliert werden, in erster Linie vom Objekttyp und in zweiter Linie vom alphabetischen Namen des Objekts ab. Die Reihenfolge der Objekttypen beim Kompilieren ist: GDAs, LDAs, PDAs, Functions, Subprogramme, externe Subroutinen, Helproutinen, Maps, Adapter, Programme, Klassen. Bei Objekten desselben Typs bestimmt die alphabetische Reihenfolge der Namen die Abfolge, in der sie katalogisiert werden. Wie zuvor erwähnt werden die Parameter des des Objekt aufrufenden Statements gegen die kompilierte Form des aufgerufenen Subprogramms geprüft. Wenn das rufende Objekt (das momentan kompiliert wird und das Objekt aufrufende Statements enthält) vor dem aufgerufenen Objekt katalogisiert wird, kann das `PCHECK`-Ergebnis falsch sein, falls die Parameter im `CALLNAT`-Statement und im aufgerufenen Subprogramm geändert wurden. Das hat zur Folge, dass das *neue* Parameterlayout im Objekt aufrufenden Statement mit dem *alten* Parameterlayout im `DEFINE DATA PARAMETER`-Statement des aufgerufenen Subprogramms verglichen wird. Lösung: ● Setzen Sie die Compiler-Option auf `PCHECK=OFF`. ● Führen Sie mit dem `CATALL`-Kommando eine generelle Kompilierung in der gesamten Library durch oder, wenn ein einzelnes oder nur wenige Objekte geändert wurden, führen Sie eine separate Kompilierung dieser Objekte durch. ● Setzen Sie die Compiler-Option auf `PCHECK=ON`. ● Führen Sie mit der Funktion `PCHECK` des `CATALL`-Kommandos eine generelle Kompilierung in der gesamten Library durch.
OFF	Es erfolgt keine Parameterprüfung. Dies ist der Standardwert.

DBSHORT - Interpretation of Database Short Field Names

Interpretation von Datenbankfeld-Namen in Programmierobjekten.

Ein in einem DDM definiertes Datenbankfeld wird durch zwei Namen beschrieben:

- durch den Kurznamen mit einer Länge von zwei Zeichen, der von Natural für die Kommunikation mit der Datenbank (insbesondere mit Adabas) verwendet wird;
- durch den Langnamen mit einer Länge von 3 bis 32 Zeichen (1 bis 32 Zeichen, wenn die darunter liegende Datenbank DB2/SQL ist), der zum Referenzieren des Feldes im Natural-Programmcode verwendet werden soll.

Unter speziellen Bedingungen können Sie in einem Natural-Programm ein Datenbankfeld mit seinem Kurznamen statt mit dem Langnamen referenzieren. Dies gilt, falls Sie Natural im Reporting Mode ohne Natural Security benutzen und falls das für den Datenbankzugriff verwendete Statement statt einer Referenz auf ein View eine Referenz auf ein DDM enthält.

Die Entscheidung, ob ein Feldname als eine Kurznamen-Referenz betrachtet wird, erfolgt abhängig von der Länge des Namens. Wenn die Feldkennung (Identifier) aus zwei Zeichen besteht, wird davon ausgegangen, dass eine Kurznamen-Referenz vorliegt; ein Feldname mit einer anderen Länge wird als Langnamen-Referenz betrachtet. Diese standardmäßige Interpretation können Sie zusätzlich beeinflussen, indem Sie die Compiler-Option `DBSHORT` auf `ON` oder `OFF` setzen:

ON	Die Verwendung eines Kurznamens zum Referenzieren eines Datenbankfeldes ist erlaubt, jedoch wird die Verwendung eines Datenbank-Kurznamens *nicht* generell gestattet (selbst wenn `DBSHORT=ON`) • für die Definition eines Feldes, wenn ein View erstellt wird; • wenn im Programmcode ein View verwendet wird; • wenn im Programmcode ein `DEFINE DATA LOCAL`-Statement zur Definition von Variablen verwendet wird; • wenn Natural Security aktiv ist. Dies ist der Standardwert.
OFF	Ein Datenbankfeld kann nur über seinen Langnamen referenziert werden. Jede Datenbankfeldkennung wird unabhängig von ihrer Länge als Langnamen-Referenz betrachtet. Wenn ein zwei Zeichen langer Name geliefert wird, der nur als Kurzname und nicht als Langname gefunden werden kann, wird bei der Kompilierung ein Syntaxfehler NAT0981 ausgegeben. Dies ermöglicht die Verwendung von Langnamen, die in einer DDM mit einer Kennungslänge von 2 Byte definiert wurden. Diese Option ist wichtig, wenn die darunter liegende Datenbank, auf die Sie mit diesem DDM zugreifen wollen, vom Typ SQL (DB2) ist und wenn dort Tabellenspalten mit einem zwei Zeichen langen Namen existieren. Dagegen wird bei allen anderen Datenbanktypen (zum Beispiel Adabas) jeder Versuch, ein Langnamenfeld mit einer Namenslänge von 2 Bytes zu definieren bei der DDM-Generierung zurückgewiesen. Darüber hinaus wird das Programm, wenn keine Kurznamenreferenzen verwendet werden (was durch `DBSHORT=OFF` erzwungen werden kann), unabhängig davon, ob es unter Natural Security kompiliert wird oder nicht.

Beispiele:

Gegeben sei die folgende Datenbankfelddefinition in dem DDM `EMPLOYEES`:

Kurzname	Langname
AA	PERSONNEL-ID

Beispiel 1:

```
OPTIONS DBSHORT=ON
READ EMPLOYEES
  DISPLAY AA      /* Datenbankfeldkurzname AA ist erlaubt
END
```

Beispiel 2:

```
OPTIONS DBSHORT=OFF
READ EMPLOYEES
  DISPLAY AA      /* Syntaxfehler NAT0981, weil DBSHORT=OFF
END
```

Beispiel 3:

```
OPTIONS DBSHORT=ON
DEFINE DATA LOCAL
1 V1 VIEW OF EMPLOYEES
  2  PERSONNEL-ID
END-DEFINE
READ V1 BY PERSONNEL-ID
  DISPLAY AA     /* Syntaxfehler NAT0981, weil PERSONNEL-ID im View definiert ist;
                 /* (selbst wenn DBSHORT=ON)
END-READ
END
```

PSIGNF - Internal Representation of Positive Sign of Packed Numbers

Interne Darstellung des positiven Vorzeichens von gepackten Zahlen.

ON	Das positive Vorzeichen einer gepackten Zahl wird intern als H’F’ dargestellt. Dies ist der Standardwert.
OFF	Das positive Vorzeichen einer gepackten Zahl wird intern als H’C’ dargestellt.

TSENABL - Applicability of TS Profile Parameter

Diese Option bestimmt, ob der Profilparameter TS für Natural-System-Libraries (d.h. für Libraries, deren Namen mit SYS beginnt, außer SYSTEM) gilt oder auch für alle Benutzer-Libraries.

Natural-Objekte, die mit der Einstellung TSENABL=ON katalogisiert werden, bestimmen die Verwendung des TS-Parameters auch dann, wenn sie sich nicht in einer System-Library befinden.

ON	Der Profilparameter TS gilt für alle Libraries.
OFF	Der Profilparameter TS gilt nur für Natural-System-Libraries. Dies ist der Standardwert.

GFID - Generation of Global Format IDs

Mit dieser Option können Sie Naturals interne Generierung von Global Format IDs steuern, um damit das Leistungsverhalten von Adabas bei der Wiederverwendbarkeit von Format-Buffer-Übersetzungen zu beeinflussen.

ON	Global Format IDs werden für alle Views generiert. Dies ist der Standardwert.
VID	Global Format IDs werden nur für in Local/Global Data Areas definierte Views generiert, aber nicht für innerhalb von Programmen definierte Views.
OFF	Es werden keine Global Format IDs generiert.

Weitere Informationen zu Global Format IDs siehe Adabas-Dokumentation.

Regeln für die Generierung von GLOBAL FORMAT-IDs in Natural

- **Für Natural-Nukleus-interne Systemdateiaufrufe:**

```
GFID=abccddee
```

dabei steht	für
a	x’F9’
b	x’22’ oder x’21’ in Abhängigkeit vom DB-Statement
cc	physische Datenbanknummer (2 Bytes)
dd	physische Dateinummer (2 Bytes)
ee	zur Laufzeit generierte Nummer (2 Bytes)

- **Für Benutzerprogramme oder Natural Utilities:**
  -

```
GFID=abbbbbbc
```

Für Dateinummern kleiner als oder gleich 255 und Adabas Version kleiner als 6.2 (siehe NTDB-Makro).

dabei steht	für
a	x’F8’ oder x’F7’ oder x’F6’
bbbbbb	Bytes 1-6 des STOD-Werts
c	physische Dateinummer

  -

```
GFID=axbbbbbc
```

Für Dateinummern über 255 und Adabas Version kleiner als 6.2.

dabei steht	für
a	x’F8’ oder x’F7’ oder x’F6’
x	physische Dateinummer - high order byte
bbbbb	Bytes 2-6 des STOD-Werts
c	physische Dateinummer - low order byte

-

```
GFID=abbbbbb
```

Für Adabas Version 6.2 oder höher.

dabei steht	für
a	x’F8’ oder x’F7’ oder x’F6’ wobei: F6=UPDATE SAME F7=HISTOGRAM F8=alle übrigen
bbbbbbb	Bytes 1-7 des STOD-Werts

Anmerkung:
STOD ist der Rückmeldewert der Store Clock Machine Instruction (STCK).

LOWSRCE - Allow Lower-Case Source

Diese Option unterstützt die Verwendung von Programmen in Sourceform mit Kleinschreibung oder gemischter Schreibweise auf Großrechnerplattformen und erleichtert so die Übertragung von derartigen Programmen von anderen Plattformen in eine Großrechnerumgebung.

ON	Gestattet jede Art von Zeichen in Klein-/Großschreibung in Source-Programmen.
OFF	Gestattet nur Großschreibung. Das bedeutet, dass Schlüsselwörter, Variablennamen und Identifier in Großbuchstaben definiert werden müssen. Dies ist der Standardwert.

Wenn Sie Zeichen in Kleinschreibung mit `LOWSRCE=ON` verwenden, müssen Sie folgendes berücksichsichtigen:

- Die Syntaxregeln für Variablennamen erlauben es, ab der zweiten Stelle Zeichen in Kleinschreibung zu benutzen. Sie können deshalb zwei Variablen gleichen Namens definieren, indem Sie die eine in Kleinschreibung und die andere in Großschreibung angeben, zum Beispiel:

```
DEFINE DATA LOCAL
1 #Vari (A20)
1 #VARI (A20)
```

Mit LOWSRCE=OFF werden die im Beispiel angegebenen Variablen als zwei verschiedene Variablen behandelt.

Mit LOWSRCE=ON unterscheidet der Compiler nicht zwischen Klein- und Großschreibung. Dies führt zu einem Syntaxfehler, weil eine doppelte Namensvergabe bei Variablen nicht erlaubt ist.

- Bei Verwendung des Session-Parameters EM (Edit Mask) in einem I/O-Statement oder in einem MOVE EDITED-Statement gibt es Zeichen, die das Daten-Layout, das einer Variablen (EM-Steuerzeichen) zugeordnet ist, und Zeichen, die Textfragmente in das Datenfeld einfügen.

  Beispiel:

```
#VARI :='1234567890'
WRITE #VARI (EM=XXXXXxxXXXXX)
```

  Bei LOWSRCE=OFF ist die Ausgabe 12345xx67890, weil bei Variablen im Alpha-Format nur X-, H- und Zirkumflex-(^)-Zeichen in Großschreibung verwendet werden können.

  Bei LOWSRCE=ON ist die Ausgabe 1234567890, weil das Zeichen x wie der Großbuchstabe C behandelt und deshalb als EM-Steuerzeichen für dieses Feldformat interpretiert wird. Um dieses Problem zu vermeiden, sollten Sie Textkonstantenfragmente in Apostrophe (') einschließen.

  Beispiel:

```
WRITE #VARI(EM=XXXXX'xx'XXXXX)
```

  Das Textfragment wird, unabhängig von den LOWSRCE-Einstellungen, *nicht* als ein EM-Steuerzeichen betrachtet.

- Da bei der Einstellung LOWSRCE=ON alle Variablennamen in Großbuchstaben umgesetzt werden, ist die Anzeige von Variablennamen in I/O-Statements (INPUT, WRITE oder DISPLAY) unterschiedlich.

  Beispiel:

```
 MOVE 'ABC' to #Vari
DISPLAY #Vari
```

  Bei LOWSRCE=OFF ist die Ausgabe:

```
#Vari -------------------- ABC
```

  Bei LOWSRCE=ON ist die Ausgabe:

```
#VARI -------------------- ABC
```

TQMARK - Translate Quotation Mark

Mit dieser Option können Sie Anführungszeichen (") in Apostrophe (') umsetzen.

ON	Jedes Anführungszeichen (") in einer Textkonstante wird als Apostroph (’) ausgegeben. Dies ist der Standardwert.
OFF	Anführungszeichen (") in einer Textkonstante werden nicht umgesetzt, sondern als Anführungszeichen beibehalten.

Beispiel:

```
RESET A (A5) A:= ’AB"CD’
WRITE ’12"34’ / A / A (EM=H(5))
END
```

Mit TQMARK ON sieht die Ausgabe so aus:

```
 12’34 AB’CD C1C27DC3C4
```

Mit TQMARK OFF sieht die Ausgabe so aus:

```
12"34 AB"CD C1C27FC3C4
```

THSEP - Dynamic Thousands Separator

Mit dieser Option können Sie beim Kompilieren die Verwendung des Tausender-Trennzeichens ermöglichen oder unterdrücken. Siehe auch Profilparameter THSEP und Profil- und Session-Parameter THSEPCH sowie Abschnitt *Trennzeichen-Angaben standardisieren* (im *Leitfaden zur Programmierung*).

ON	Das Tausender-Trennzeichen wird verwendet. Jedes Tausender-Trennzeichen, das nicht Teil eines Zeichenketten-Literals ist, wird intern durch ein Steuerzeichen ersetzt.
OFF	Das Tausender-Trennzeichen wird nicht verwendet, d.h. der Compiler erzeugt kein Steuerzeichen für Tausender-Trennzeichen. Dies ist die Kompatibilitätseinstellung.

CPAGE - Code Page Support for Alphanumeric Constants

(Codepage-Unterstützung bei alphanumerischen Konstanten)

Mit der CPAGE-Option können Sie eine Konvertierungsroutine aktivieren, die, wenn das Objekt zur Laufzeit gestartet wird, alle alphanumerischen Konstanten von der Codepage, die bei der Kompilierung aktiv war, in die Codepage, die zur Laufzeit aktiv ist, umsetzt.

ON	Codepage-Unterstützung für alphanumerische Zeichenketten ist eingeschaltet.
OFF	Codepage-Unterstützung für alphanumerische Zeichenketten ist ausgeschaltet. Dies ist der Standardwert.

DB2ARRY - Support DB2 Arrays in SQL SELECT and INSERT Statements

(Unterstützung von DB2-Arrays in SQL SELECT- und INSERT-Statements)

Mit der Option DB2ARRY können Sie die Recherche und/oder das Einfügen in mehreren Reihen in DB2 mit nur einer Ausführung eines SQL SELECT- oder INSERT-Statements aktivieren. Dies ermöglicht die Angabe von Arrays als Zielfelder im SQL SELECT-Statement oder als Quellfelder im SQL

INSERT-Statement. Wenn DB2ARRY auf ON gesetzt ist, ist es nicht mehr möglich, alphanumerische Natural-Arrays für DB2 VARCHAR/GRAPHIC-Spalten zu verwenden. Stattdessen müssen Sie lange alphanumerische Natural-Variablen verwenden.

ON	DB2-Array-Unterstützung ist eingeschaltet.
OFF	DB2-Array-Unterstützung ist ausgeschaltet. Dies ist der Standardwert.

CHKRULE - Validierung von INCDIR-Statements in Maps

Mit der Option CHKRULE können Sie eine während des Katalogisierungsvorgangs für Maps stattfindende Validierungsprüfung ein- bzw. ausschalten.

ON	Die INCDIR-Validierung ist eingeschaltet. Wenn die bzw. das im INCDIR-Kontrollstatement referenzierte Datei (DDM) bzw. Feld nicht existiert, wird zur Kompilierungszeit ein Syntaxfehler (NAT0721) ausgegeben. Beim Anlegen einer Map können Sie Felder einfügen, die schon in einem anderen existierenden Programmierobjekt definiert sind. Das funktioniert bei fast allen Arten von Objekten, in denen Sie Variablen definieren können, und ebenso bei DDMs. Wenn es sich bei dem eingefügten Feld um eine Datenbankvariable handelt, fügt der Map Editor automatisch (neben dem eingefügten Feld) ein INCDIR-Statement in den Map-Statement-Rumpf ein, um damit das Laden und die Übernahme einer Predict-Regel auszulösen, wenn die Map (per STOW-Befehl) kompiliert wird. Die Funktionsweise ist ähnlich dem Vorgang bei der Verarbeitung eines INCLUDE-Statements. Anstatt jedoch Sourcecode-Zeilen aus einem Copycode-Objekt zu übernehmen, werden sie in diesem Fall aus Predict übergeben. Als Suchschlüssel zum Auffinden der Regel(n) dienen der DDM-Name (der als der Feld-Name betrachtet wird) und der Feld-Name. Beide werden im INCDIR-Statement angegeben. Eine während der Kompilierung angeforderte INCDIR-Regel muss aber nicht zwangsläufig in Predict gefunden werden, weil für ihr Vorhandensein durchaus keine Notwendigkeit besteht. Das bedeutet, dass keine Fehlersituation vorliegt, wenn eine gesuchte Regel nicht gefunden wird. Bei Übernahme von Feldern aus einer DDM in eine Map, werden entsprechende INCDIR-Statements erzeugt, die den aktuelle DDM- und Feldnamen als "Suchschlüssel" enthalten, mit dem eventuell existierende Regeln aus Predict abgerufen werden. Falls aber die DDM nach dem Kopiervorgang umbenannt wird, bleibt der alte (nicht mehr gültige) DDM-Name im INCDIR-Statement erhalten, was zur Folge hat, dass keine Regel geladen und der Programmierer darüber nicht in Kenntnis gesetzt wird. Eine solche Situation tritt aber nicht nur im Falle einer DDM-Umbenennung auf. Wahrscheinlicher ist, dass sie durch versehentliche Zuordnung einer falschen FDIC-Datei verursacht wird. In diesem Fall ist der DDM-Name zwar gültig, kann aber in der aktuellen Predict-Systemdatei nicht gefunden werden. Das Ergebnis ist das gleiche wie wenn die DDM überhaupt nicht existiert, das heißt, die erwartungsgemäß aus Predict hinzuzufügenden Verarbeitungsregeln werden nicht eingefügt.
OFF	Die INCDIR-Validierung ist ausgeschaltet. Dies ist der Standardwert.

Kompilierungsoptionen für Versionskompatibilität (Remote Mainframe-Umgebung)

Folgende Optionen stehen zur Verfügung:

- FINDMUN - Detect Inconsistent Comparison Logic in FIND Statements
- MASKCME - MASK Compatible with MOVE EDITED
- NMOVE22 - Assignment of Numeric Variables of Same Length and Precision
- V41COMP - Disable New Version 4.2 Syntax

Diese Optionen entsprechen den gleichnamigen Schlüsselwortparametern des Profilparameters `CMPO` bzw. des Parameter-Makros `NTCMPO`.

FINDMUN - Detect Inconsistent Comparison Logic in FIND Statements

Mit Natural Version 2.3 hat sich die Vergleichslogik für multiple Felder in der `WITH`-Klausel des `FIND`-Statements geändert.

Das hat zur Folge, dass Sie andere Ergebnisse erhalten, wenn Sie Version-2.3-Programme, die bestimmte Formen von `FIND`-Statements enthalten, unter Version 2.3 kompilieren. Mit dieser Option können Sie nach `FIND`-Statements suchen, deren `WITH`-Klausel multiple Felder in einer Weise verwendet, die nicht mehr mit der verbesserten Vergleichslogik von Version 3.1 konsistent ist.

ON	Fehler NAT0998 wird für jedes bei der Kompilierung entdeckte derartige `FIND`-Statement ausgegeben.
OFF	Es erfolgt keine Suche nach derartigen `FIND`-Statements. Dies ist der Standardwert.

Die Vergleichslogik für multiple Felder in der `WITH`-Klausel des `FIND`-Statement wurde ab Natural Version 2.3 so geändert, dass sie mit der Vergleichslogik bei anderen Statements (z.B. `IF`) konsistent ist.

Man kann vier verschiedene Formen des `FIND`-Statement unterscheiden. Dabei wird davon ausgegangen, dass das Feld `MU` in den folgenden Beispielen ein multiples Feld ist.

1.

```
FIND
XYZ-VIEW WITH MU = 'A'
```

   Bei Version 2.2 und höher liefert dieses Statement Datensätze zurück, in denen mindestens eine Ausprägung von `MU` den Wert `A` hat.

2.

```
FIND XYZ-VIEW WITH
MU NOT EQUAL 'A'
```

   Bei Version 2.2 liefert dieses Statement Datensätze zurück, in denen keine Ausprägung von `MU` den Wert `A` hat (wie bei Punkt 4). Bei Version 2.3 und höher liefert dieses Statement Datensätze zurück, in denen mindestens eine Ausprägung von `MU` nicht den Wert `A` hat.

3.

```
FIND XYZ-VIEW WITH NOT MU NOT EQUAL
'A'
```

Bei Version 2.2 liefert dieses Statement Datensätze zurück, in denen *mindestens eine Ausprägung* von MU den Wert A hat (wie bei Punkt 1). Bei Version 2.3 und höher liefert dieses Statement Datensätze zurück, in denen *jede Ausprägung* von MU den Wert A hat.

4.

```
FIND XYZ-VIEW WITH NOT MU
= 'A'
```

Bei Version 2.2 und höher liefert dieses Statement Datensätze zurück, in denen *keine Ausprägung* von MU den Wert A hat. Das bedeutet, wenn Sie vorhandene Version-2.2-Programme, die FIND-Statements der Formen 2 und 3 enthalten, unter der Version 2.3 kompilieren, liefern diese unterschiedliche Ergebnisse.

Wenn Sie die Option FINDMUN=ON benutzen, wird bei jedem beim Kompilieren festgestellten FIND-Statement der Form 2 oder 3 ein Fehler (NAT0998) ausgegeben.

Wenn Sie in diesen Fällen weiterhin dieselben Ergebnisse wie bei der Version 2.2 erhalten möchten, müssen Sie diese Statements wie folgt schreiben:

Bei Form 2:

```
FIND XYZ-VIEW WITH MU NOT EQUAL 'A'
```

ändern in

```
FIND XYZ-VIEW WITH NOT MU = 'A'
```

Bei Form 3:

```
FIND XYZ-VIEW WITH NOT MU NOT EQUAL 'A'
```

ändern in

```
FIND XYZ-VIEW WITH MU = 'A'
```

MASKCME - MASK Compatible with MOVE EDITED

Mit dieser Option können Sie die MASK-Option mit dem MOVE EDITED-Statement kompatibel machen.

ON	Der Bereich der gültigen Jahreswerte, die zu den YYYY-Maskenzeichen passen, ist 1582 bis 2699, damit die MASK-Option mit dem MOVE EDITED-Statement kompatibel wird. Wenn der Profilparameter MAXYEAR auf 9999 gesetzt ist, erstreckt sich der Bereich der gültigen Jahreswerte von 1582 bis 9999.
OFF	Der Bereich der gültigen Jahreswerte, die zu den YYYY-Maskenzeichen passen, ist 0000 - 2699. Dies ist der Standardwert. Wenn der Profilparameter MAXYEAR auf 9999 gesetzt ist, erstreckt sich der Bereich der gültigen Jahreswerte von 0000 bis 9999.

NMOVE22 - Assignment of Numeric Variables of Same Length and Precision

ON	Die Zuordnung von numerischen Variablen, bei denen Source und Target dieselbe Länge und Genauigkeit haben, erfolgt wie bei Natural Version 2.2.
OFF	Die Zuordnung von numerischen Variablen, bei denen Source und Target dieselbe Länge und Genauigkeit haben, erfolgt wie bei Natural Version 2.3 und höher, d.h., sie werden so verarbeitet, als ob Source und Target eine unterschiedliche Länge oder Genauigkeit hätten. Dies ist der Standardwert.

V41COMP - Disable New Version 4.2 Syntax

Wichtig:
Diese Compiler-Option ist nur bei der Natural Version 4.2 vorhanden, um einen sanften Übergang zu ermöglichen. Sie wird in einem späteren Release im Anschluss an Natural Version 4.2 entfallen.

Einige der mit Natural Version 4.2 eingeführten Funktionen und Programmierungsmerkmale verursachen Probleme, wenn ein unter Version 4.1 entwickeltes und kompiliertes Programm neu kompiliert werden soll, um es in einer Version-4.1-Umgebung in Betrieb zu nehmen. Die betreffenden Funktionen und Merkmale sind nachfolgend aufgeführt.

Die Option `V41COMP` dient dazu, deratige Unverträglichkeiten zu finden und eine Fehlermeldung auszugeben, die durch einen Reason Code aussagt, warum die Neukompilierung nicht erfolgreich war.

Mögliche Werte sind:

ON	Wenn ein Programm unter Version 4.2 kompiliert wird, dann wird jeder Versuch, ein von Version 4.2, aber nicht von Version 4.1 unterstütztes Syntaxkonstrukt zu verwenden, zurückgewiesen, und es wird ein Syntaxfehler NAT0647 und ein entsprechender Reason Code ausgegeben (siehe unten).
OFF	Es erfolgt keine Prüfung auf Kompatibilität mit Version 4.1. Dies ist der Standardwert.

Kompilierungsrelevante Unterschiede zwischen Version 4.2 und 4.1

Die folgende Tabelle enthält eine Übersicht über kompilierungsrelevante Unterschiede zwischen Version 4.2 und 4.1 und zeigt den Reason Code, der ausgegeben wird, wenn eine inkompatible Syntax festgestellt wird:

Funktion oder Merkmal	Version 4.2	Version 4.1	Reason Code
Neues Format U (Unicode)	möglich	unbekannt	001
Array mit variabler Anzahl an Ausprägungen: X-array, zum Beispiel: `DEFINE DATA LOCAL 1 #ARR (A10/1:*)`	möglich	unbekannt	002
Mögliche Länge von alphanumerischen Konstanten und Literalen (Konstanten)	1 Byte - 1 GB	1 Byte - 253 Bytes (NAT0264)	003

Funktion oder Merkmal	Version 4.2	Version 4.1	Reason Code
Neue Kompilerparameter `THSEP` Tausender-Trennzeichen in Editiermasken `CPAGE` Codepage-Umsetzung für alphanumerische Konstante ermöglichen	möglich	unbekannt	004

Neue Statements
`MOVE NORMALIZED`
`MOVE ENCODED`
`PARSE XML`
`REQUEST DOCUMENT`
`EXPAND / REDUCE / RESIZE ARRAY`
	möglich	unbekannt	005

Statement `SET GLOBALS` • Session-Parameter `CPCVERR=ON/OFF` • zulässig im Structured-Modus (`SM=ON`)	möglich	unbekannt	006

Neue Systemvariablen
`*PARSE-COL`
`*PARSE-LEVEL`
`*PARSE-NAMESPACE-URI`
`*PARSE-ROW`
`*PARSE-TYPE`
`*CODEPAGE`
`*LOCALE`
`*TYPE`
`*CURRENT-UNIT`
`*UBOUND`
`*LBOUND`
	möglich	unbekannt	007

Nicht benutzt	-	-	008
Länge und Typ von Source-Parametern, die mit `INCLUDE`-Statement geliefert werden Beispiel: `INCLUDE COPY01 'WRITE *LINE' 'WRITE *PROGRAM'`	jede Länge und Format U (Unicode) erlaubt	nur alphanumerisch mit einer Länge von max. 80 Bytes	009

Funktion oder Merkmal	Version 4.2	Version 4.1	Reason Code
Definition eines Adabas LA-Feldes in einer Data View • mit Größe > 253 Bytes oder • des Typs `DYNAMIC`	möglich	unbekannt	010