Datenbereiche (Data Areas)

Ein Natural-Datenbereich (Data Area) ist ein Natural-Programmierobjekt, das von mehreren Natural-Programmen, -Subprogrammen, -Subroutinen, Helproutinen oder Klassen benutzt werden kann.

Dieses Kapitel behandelt folgende Themen:

- Verwendung von Datenbereichen
- Local Data Area
- Global Data Area
- Parameter Data Area

Verwendung von Datenbereichen

Wie im Abschnitt *Felder definieren* erläutert, müssen alle Felder, die in einem Programm verwendet werden sollen, mit einem `DEFINE DATA`-Statement definiert werden.

Die Felder können entweder innerhalb des `DEFINE DATA`-Statements selbst definiert werden oder sie können außerhalb des Programms in einem separaten Datenbereich (Data Area) definiert werden, der dann vom `DEFINE DATA`-Statement referenziert wird.

Eine solche separate Data Area ist ein Natural-Programmierobjekt, das von mehreren Natural-Programmen, -Subprogrammen, -Subroutinen, Helproutinen oder Klassen benutzt werden kann. Eine Data Area enthält Datenelement-Definitionen, wie z.B. benutzerdefinierte Variablen, Konstanten und Datenbankfelder aus einem Datendefinitionsmodul (DDM).

Alle Data Areas werden mit dem Data Area Editor erstellt und editiert.

Mit Natural können Sie folgende Arten von Data Areas anlegen und referenzieren:

- Local Data Area
- Global Data Area
- Parameter Data Area

Local Data Area

Als "local" definierte Variablen können nur von einem einzigen Natural-Programmierobjekt benutzt werden.

Sie haben zwei Möglichkeiten, lokale Daten zu definieren:

- Sie können die Daten innerhalb des Programms definieren.

- Sie können die Daten außerhalb des Programms in einem separaten Natural-Programmierobjekt, einer Local Data Area, definieren.

Ein Natural-Programmierobjekt des Typs Local Data Area (LDA) ermöglicht es Programmen, Subprogrammen, externen Subroutinen und Klassen identische Datenelement-Definitionen zu benutzen (z.B. identische Feldnamen und Formate), aber den Inhalt der Datenelemente für jedes einzelne Objekt separat zu halten.

Eine Local Data Area wird initialisiert, wenn ein Programm, Subprogramm oder eine Klasse oder externe Subroutine, das oder die diese Local Data Area benutzt, ausgeführt wird.

Im ersten Beispiel sind die Felder innerhalb des `DEFINE DATA`-Statements des Programms definiert. Im zweiten Beispiel sind dieselben Felder in einer Local Data Area definiert, und das `DEFINE DATA`-Statement enthält lediglich eine Referenz auf diese Data Area.

Beispiel 1 — Felddefinitionen innerhalb des DEFINE DATA-Statements:

```
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 PERSONNEL-ID
1 #VARI-A (A20)
1 #VARI-B (N3.2)
1 #VARI-C (I4)
END-DEFINE
...
```

Beispiel 2 — Felddefinitionen in einer separaten Data Area:

Programm:

Das Programm selbst enthält keine Felddefinitionen, sondern referenziert die in der `LDA39` enthaltenen Felddefinitionen.

```
DEFINE DATA LOCAL
  USING LDA39
END-DEFINE
  ...
```

Referenzierte Local Data Area `LDA39`:

```
I T L  Name                            F Length     Miscellaneous
All -- ------------------------------- - ---------- ------------------------>
  V  1 VIEWEMP                                      EMPLOYEES
     2 PERSONNEL-ID                    A          8
     2 FIRST-NAME                      A         20
     2 NAME                            A         20
     1 #VARI-A                         A         20
     1 #VARI-B                         N        3.2
     1 #VARI-C                         I          4
```

Tipp: Um eine übersichtlich strukturierte und einheitliche Anwendung zu erhalten, ist es in der Regel besser, Felder in Data Areas außerhalb der Programme zu definieren.

Global Data Area

Wenn Sie die Datenelemente in einem separaten Bereich definieren möchten, der von mehreren Natural-Programmierobjekten genutzt werden kann, verwenden Sie eine Globa Data Area.

Die folgenden Themen werden erörtert:

- GDA anlegen und referenzieren
- GDA-Instanzen
- Datenblöcke

GDA anlegen und referenzieren

GDAs werden mit dem Natural Data Area Editor angelegt und geändert. Weitere Informationen entnehmen Sie dem Abschnitt *Data Area Editor* in der *Editors*-Dokumentation.

Eine GDA, die von einem Natural-Programmierobjekt referenziert wird, muss in derselben Natural-Library (oder in einer für diese Library definierten Steplib) gespeichert werden, in der auch das diese GDA referenzierende Objekt gespeichert ist.

Wenn eine GDA mit Namen `COMMON` in einer Library vorhanden ist, wird das Programm mit Namen `ACOMMON` automatisch aufgerufen, wenn Sie sich mit einem `LOGON` in dieser Library anmelden.

Wichtig:
Wenn Sie eine Anwendung erstellen, bei der mehrere Natural-Programmierobjekte eine GDA referenzieren, denken Sie bitte daran, dass Änderungen an den Datenelement-Definitionen in der GDA alle Natural-Programmierobjekte betreffen, die diese Data Area referenzieren. Deshalb müssen diese Objekte mittels des Kommandos `CATALOG` oder `STOW` neu kompiliert werden, nachdem die GDA geändert worden ist.

Um eine GDA zu benutzen, muss ein Natural-Programmierobjekt sie mit der `GLOBAL`-Klausel des `DEFINE DATA`-Statements referenzieren.

Jedes Natural-Programmierobjekt kann nur eine GDA referenzieren; d.h. ein `DEFINE DATA`-Statement darf nicht mehr als eine `GLOBAL`-Klausel enthalten.

GDA-Instanzen

Die erste Instanz einer GDA wird angelegt und zur Laufzeit initialisiert, wenn das erste, sie referenzierende Natural-Programmierobjekt ausgeführt wird.

Sobald eine GDA-Instanz angelegt worden ist, können die Datenwerte, die sie enthält, von allen Natural-Programmierobjekten gemeinsam benutzt werden, die diese GDA referenzieren (`DEFINE DATA GLOBAL`-Statement) und die von einem `PERFORM`-, `INPUT`- oder `FETCH`-Statement aufgerufen werden. Alle Objekte, die eine GDA-Instanz gemeinsam benutzen, greifen auf dieselben Datenelemente zu.

Eine neue GDA-Instanz wird erstellt, wenn Folgendes gilt:

- Ein Subprogramm, das eine GDA (eine *beliebige* GDA) referenziert, wird mit einem CALLNAT-Statement aufgerufen.
- Ein Subprogramm, das *keine* GDA referenziert, ruft ein Programmierobjekt auf, das eine GDA (eine *beliebige* GDA) referenziert.

Beim Anlegen einer neuen Instanz einer GDA wird die aktuelle GDA-Instanz zeitweilig unterbrochen und die Datenwerte, die sie enthält, werden in den Stack geschrieben. Das Subprogramm referenziert dann die Datenwerte in der neu erstellten GDA-Instanz. Auf die Datenwerte in der/den zeitweilig unterbrochenen GDA-Instanz/Instanzen ist kein Zugriff möglich.

Ein Programmierobjekt bezieht sich nur auf eine GDA-Instanz und kann nicht auf vorherige GDA-Instanzen zugreifen. Ein GDA-Datenelement kann nur an ein Subprogramm übergeben werden, wenn das Element als ein Parameter im CALLNAT-Statement definiert wird.

Wenn das Subprogramm zum aufrufenden Programmierobjekt zurückkehrt, wird die es referenzierende GDA-Instanz gelöscht, und die vorher zeitweilig unterbrochene GDA-Instanz wird mit ihren Datenwerten wieder aufgenommen.

Eine GDA-Instanz und ihr Inhalt wird gelöscht, wenn einer der folgenden Punkte gilt:

- Das nächste LOGON wird ausgeführt.
- Eine andere GDA wird auf derselben Stufe referenziert (Stufen sind später in diesem Abschnitt beschrieben).
- Ein RELEASE VARIABLES-Statement wird ausgeführt.

  In diesem Fall werden die Datenwerte in einer GDA-Instanz zurückgesetzt, und zwar entweder wenn ein Programm auf der Stufe 1 seine Ausführung beendet, oder wenn das Programm ein anderes Programm über ein FETCH- oder RUN-Statement aufruft.

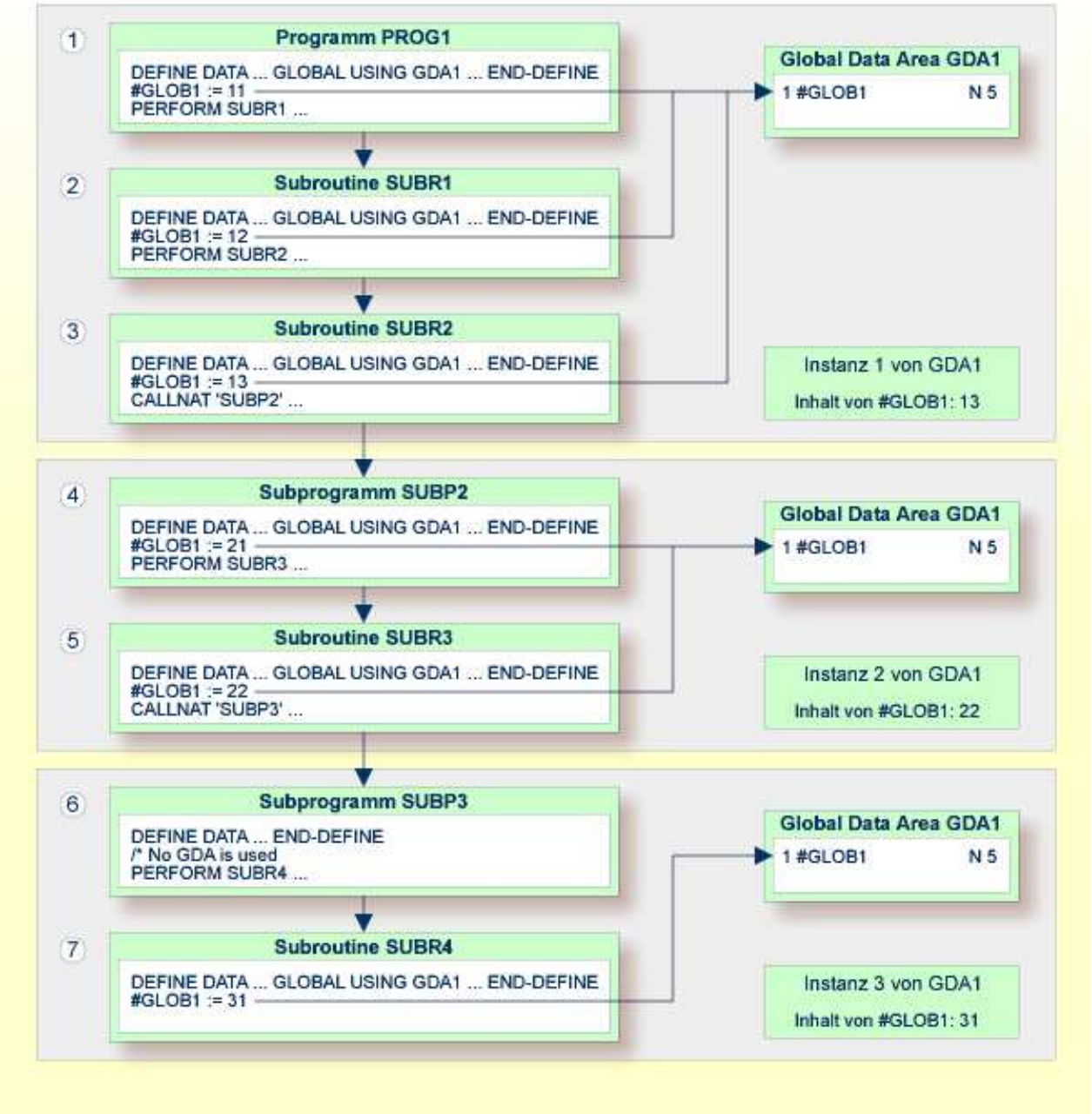
  Die folgende Grafik zeigt, wie Programmierobjekte GDAs referenzieren und Datenelemente in GDA-Instanzen mitbenutzen.

GDA-Instanzen gemeinsam benutzen

Die Grafik weiter unten veranschaulicht, dass ein eine GDA referenzierendes Subprogramm die Datenwerte in einer GDA-Instanz nicht gemeinsam benutzen kann, die von dem aufrufenden Programm referenziert wird.

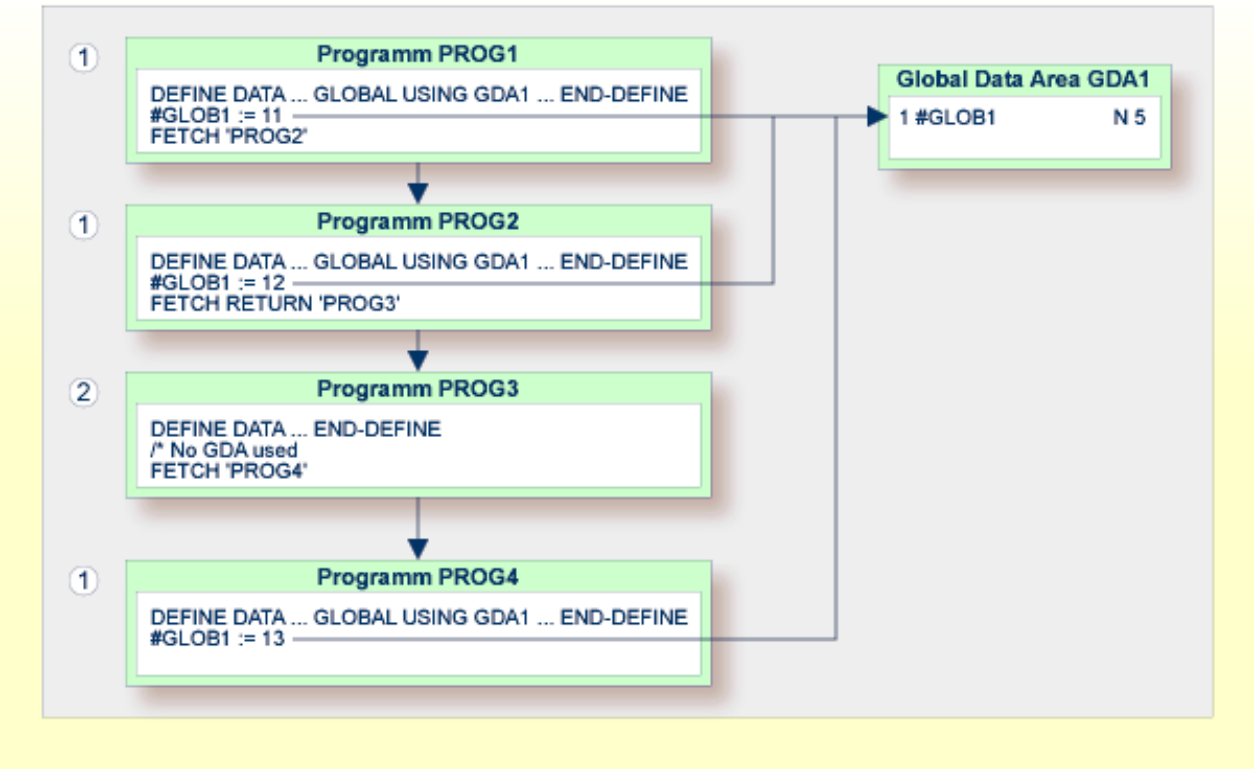
Ein Subprogram, das dieselbe GDA referenziert wie das aufrufende Programm, erstellt eine neue Instanz dieser GDA. Die Datenelemente, die in einer GDA definiert sind, die von einem Subprogramm referenziert wird, können aber von einer vom Subprogramm aufgerufenen Subroutine oder Helproutine gemeinsam benutzt werden.

Die folgende Grafik zeigt drei GDA-Instanzen von GDA1 und die Endwerte, die jeder GDA-Instanz vom Datenelement #GLOB1 zugewiesen werden. Die Zahlen ① bis ⑦ verweisen auf die hierarchischen Stufen der Programmierobjekte.



Die folgende Grafik veranschaulicht, dass Programme, die dieselbe GDA referenzieren und sich gegenseitig mit dem `FETCH`- oder `FETCH RETURN`-Statement aufrufen, die in dieser GDA definierten Datenelemente gemeinsam benutzen. Wenn eines dieser Programme keine GDA referenziert, bleibt die Instanz der vorher referenzierten GDA aktiv, und die Werte der Datenelemente werden zurückbehalten.

Die Ziffern (1) und (2) verweisen auf die hierarchischen Stufen der Programmierobjekte.



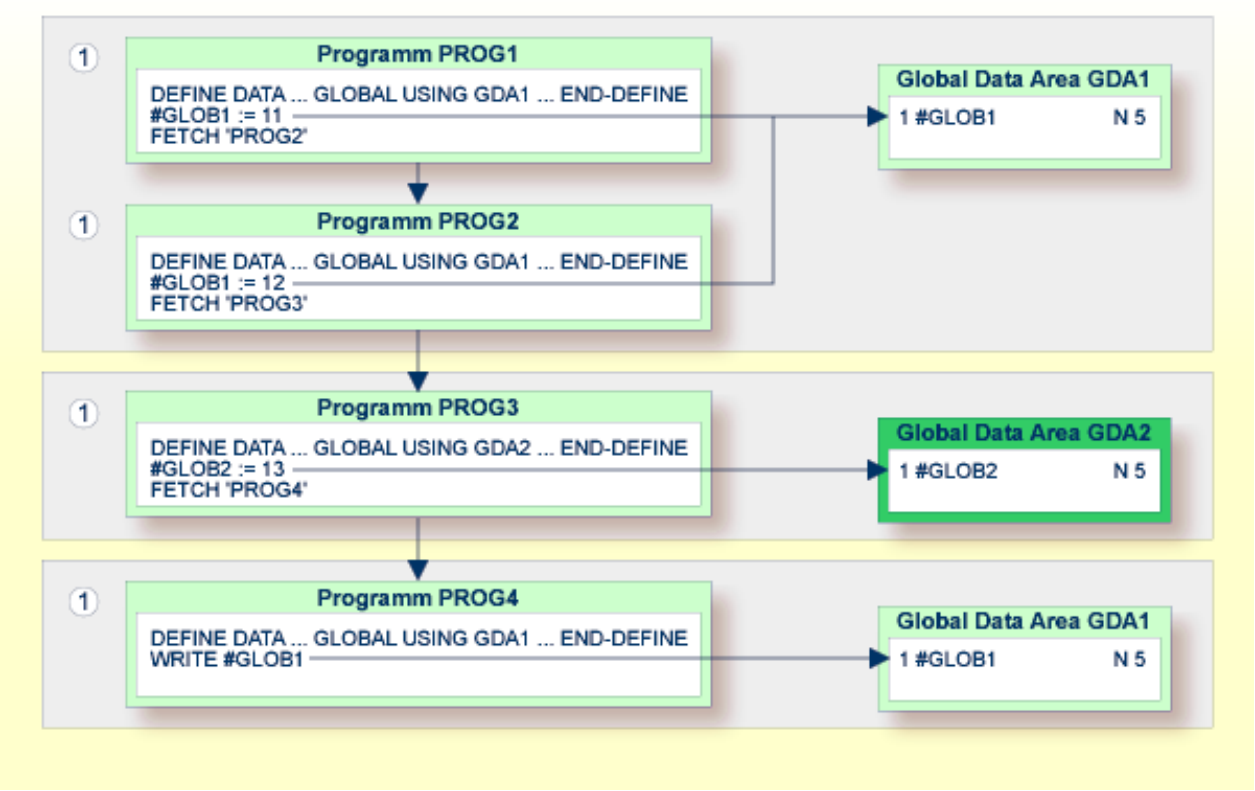
Die folgende Grafik veranschaulicht, dass, wenn ein Programm das `FETCH`-Statement benutzt, um ein anderes Programm aufzurufen, das eine unterschiedliche GDA referenziert, wird die aktuelle, vom aufrufenden Programm referenzierte Instanz der GDA (hier: GDA1) gelöscht. Wenn diese GDA dann erneut von einem anderen Programm referenziert wird, eine neue Instanz dieser GDA erstellt wird, bei der alle Datenelemente ihre Anfangswerte haben.

Sie können das `FETCH RETURN`-Statement nicht benutzen, um ein anderes Programm aufzurufen, das eine unterschiedliche GDA referenziert.

Die Ziffer (1) verweist auf die hierarchische Stufe der Programmierobjekte.

Die aufrufenden Programme `PROG3` und `PROG4` beeinflussen die GDA-Instanzen wie folgt:

- Das Statement `GLOBAL USING GDA2` in `PROG3` erstellt eine Instanz von `GDA2` und löscht die aktuelle Instanz von `GDA1`.
- Das Statement `GLOBAL USING GDA1` in PROG4 löscht die aktuelle Instanz von `GDA2` und erstellt eine neue Instanz von `GDA1`. Als Ergebnis davon zeigt das `WRITE`-Statement den Wert Null (0) an.



Datenblöcke

Um Datenspeicher zu sparen, können Sie eine GDA mit Datenblöcken erstellen.

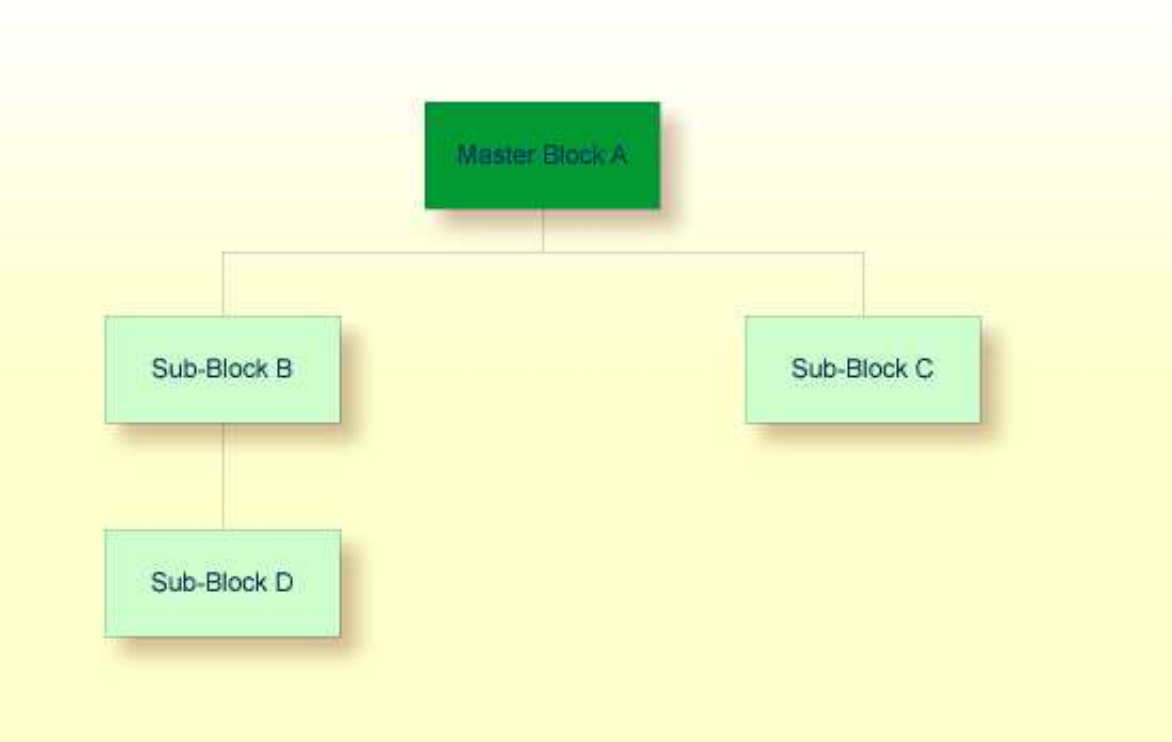
Folgende Themen werden in diesem Abschnitt behandelt:

- Beispiel für die Benutzung von Datenblöcken
- Datenblöcke definieren
- Block-Hierarchien

Beispiel für die Benutzung von Datenblöcken

Datenblöcke können sich bei der Programmausführung gegenseitig überlagern, was zu einem Einsparen von Speicherplatz führt.

Gehen wir beispielsweise davon aus, dass bei der folgenden vorgegebenen Hierarchie den Blöcken B und C derselbe Speicherbereich zugewiesen würde. So wäre es nicht möglich, dass die Blöcke B und C gleichzeitig in Benutzung sind. Die Änderung von Block B würde zur Zerstörung des Inhalts von Block C führen.



Datenblöcke definieren

Datenblöcke werden im Data-Area-Editor definiert. Sie bauen die Block-Hierarchie auf, indem Sie angeben, welcher Block welchem anderen untergeordnet ist: geben Sie dazu den Namen des übergeordneten "Parent"-Blocks in das Kommentarfeld der Block-Definition ein.

In dem folgenden Beispiel sind SUB-BLOCKB und SUB–BLOCKC dem Block MASTER–BLOCKA untergeordnet; SUB–BLOCKD ist SUB–BLOCKB untergeordnet.

Die maximale Anzahl der Block-Stufen ist 8 (einschießlich des Master-Blocks).

Beispiel:

Global Data Area G-BLOCK:

```
I T L Name                            F Leng Index/Init/EM/Name/Comment
- - - -------------------------------- - ---- --------------------------------
  B   MASTER-BLOCKA
    1 MB-DATA01                       A   10
  B   SUB-BLOCKB                            MASTER-BLOCKA
    1 SBB-DATA01                      A   20
  B   SUB-BLOCKC                            MASTER-BLOCKA
    1 SBC-DATA01                      A   40
  B   SUB-BLOCKD                            SUB-BLOCKB
    1 SBD-DATA01                      A   40
```

Um die spezifischen Blöcke einem Programm zur Verfügung zu stellen, benutzen Sie die folgende Syntax im DEFINE DATA-Statement:

Programm 1:

```
DEFINE DATA GLOBAL
       USING G-BLOCK
       WITH MASTER-BLOCKA
END-DEFINE
```

Programm 2:

```
DEFINE DATA GLOBAL
       USING G-BLOCK
       WITH MASTER-BLOCKA.SUB-BLOCKB
END-DEFINE
```

Programm 3:

```
DEFINE DATA GLOBAL
       USING G-BLOCK
       WITH MASTER-BLOCKA.SUB-BLOCKC
END-DEFINE
```

Programm 4:

```
DEFINE DATA GLOBAL
       USING G-BLOCK
       WITH MASTER-BLOCKA.SUB-BLOCKB.SUB-BLOCKD
END-DEFINE
```

Mit dieser Struktur kann Programm 1 die Daten in MASTER–BLOCKA gemeinsam mit Programm 2, Programm 3 oder Programm 4 benutzen. Allerdings können die Programme 2 und 3 nicht dieselben Data Areas von SUB–BLOCKB und SUB–BLOCKC gemeinsam benutzen, weil diese Datenblöcke auf derselben Struktur-Stufe definiert sind, und folglich denselben Speicherbereich belegen.

Block-Hierarchien

Besonders sorgfältig müssen Sie vorgehen, wenn Sie Datenblock-Hierarchien benutzen. Gehen wir von folgendem Szenario mit drei Programmen aus, die eine Datenblock-Hierarchie verwenden:

Programm 1:

```
DEFINE DATA GLOBAL
       USING G-BLOCK
       WITH MASTER-BLOCKA.SUB-BLOCKB
END-DEFINE
*
MOVE 1234 TO SBB-DATA01
FETCH 'PROGRAM2'
END
```

Programm 2:

```
DEFINE DATA GLOBAL
       USING G-BLOCK
       WITH MASTER-BLOCKA
END-DEFINE
*
FETCH 'PROGRAM3'
END
```

Programm 3:

```
DEFINE DATA GLOBAL
       USING G-BLOCK
       WITH MASTER-BLOCKA.SUB-BLOCKB
END-DEFINE
*
WRITE SBB-DATA01
END
```

Erläuterung:

- Programm 1 benutzt die Global Data Area G–BLOCK mit MASTER–BLOCKA und SUB–BLOCKB. Das Programm ändert ein Feld in SUB–BLOCKB und ruft Programm 2 mit einem FETCH-Statement auf, welches nur MASTER–BLOCKA in seiner Datendefinition angegeben hat.
- Programm 2 setzt SUB–BLOCKB zurück (löscht den Inhalt von SUB–BLOCKB). Der Grund dafür ist, dass ein Programm auf Stufe 1 (zum Beispiel ein mit einem FETCH-Statement aufgerufenes Programm) alle Datenblöcke zurücksetzt, die den Blöcken untergeordnet sind, die es in seiner eigenen Datendefinition festlegt.
- Programm 2 ruft jetzt Programm 3 mit einem FETCH-Kommando auf, welches das in Programm 1 geänderte Feld anzeigen soll, aber es gibt einen leeren Bildschirm zurück.

Einzelheiten zu den Programmfstufen entnehmen Sie dem Abschnitt *Mehrere Stufen (Levels) aufgerufener Objekte*.

Parameter Data Area

Ein Natural-Programmierobjekt des Typs Parameter Data Area (PDA) wird benutzt, um die Datenelemente zu definieren, die als Parameter an ein Subprogramm, eine externe Subroutine oder Helproutine übergeben werden.

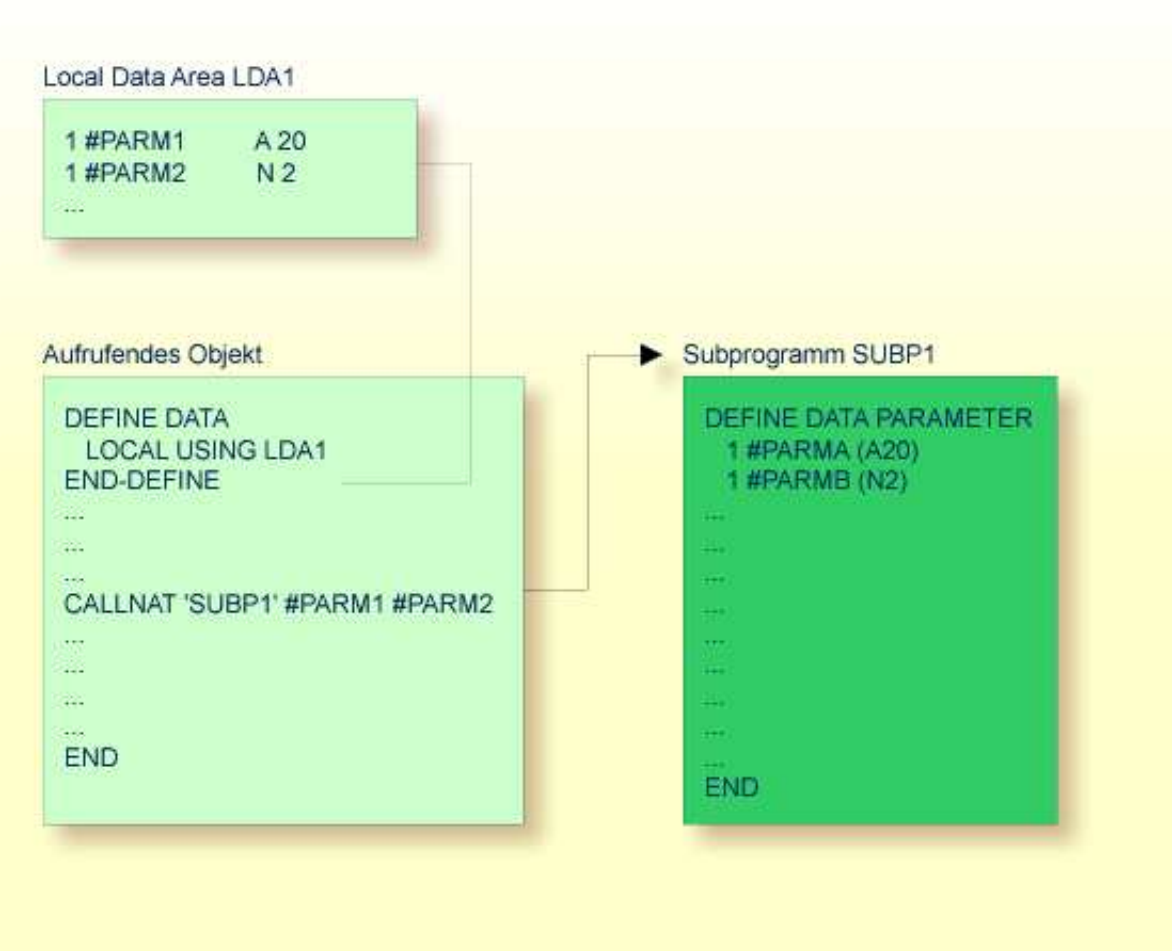
Eine PDA ermöglicht es Subprogrammen, externen Subroutinen und Helproutinen, dieselben Datenelement-Definitionen zu benutzen (zum Beispiel, identische Feldnamen und -formate).

Ein Subprogramm wird mit einem CALLNAT-Statement aufgerufen. Mit dem CALLNAT-Statement können Parameter von dem aufrufenden Objekt an das Subprogramm übergeben werden.

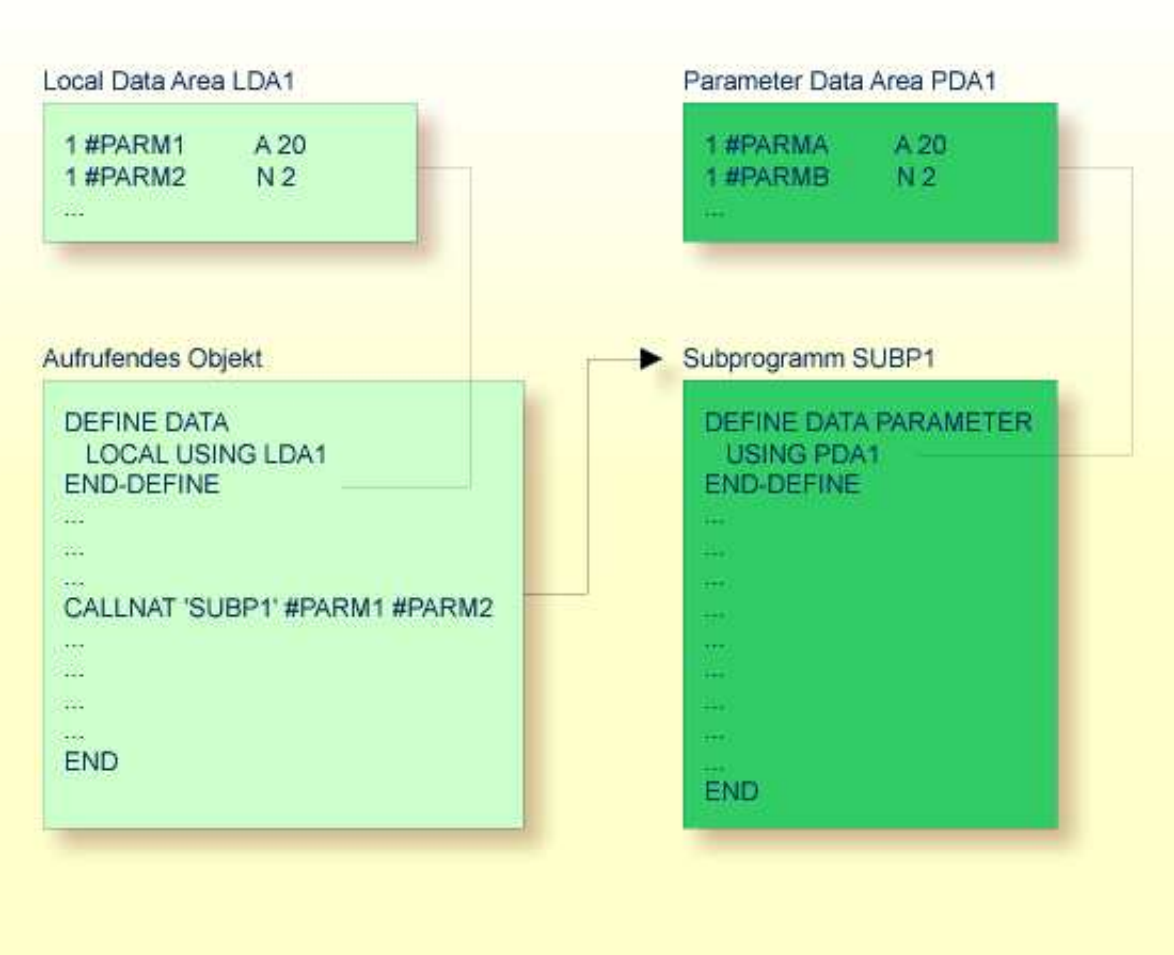
Diese Parameter müssen im Subprogramm in einem DEFINE DATA PARAMETER-Statement definiert werden:

- sie können entweder in der PARAMETER-Klausel des DEFINE DATA-Statements selbst definiert werden oder
- sie können definiert werden in einer separaten Parameter Data Area, die von dem DEFINE DATA PARAMETER-Statement referenziert wird.

Innerhalb des DEFINE DATA PARAMETER-Statements definierte Parameter



Separat in einer Parameter Data Area definierte Parameter



In der gleichen Weise wie beim `CALLNAT`-Statement müssen Parameter, die mit einem `PERFORM`-Statement an eine externe Subroutine übergeben werden, in der externen Subroutine in einem `DEFINE DATA PARAMETER`-Statement definiert werden.

Im aufrufenden Objekt müssen die an das Subprogramm bzw. die Subroutine übergebenen Parametervariablen nicht in einer Parameter Data Area definiert werden; in der obigen Abbildung sind sie in einer vom aufrufenden Objekt benutzten Local Data Area definiert (man hätte sie aber auch in einer Global Data Area definieren können).

Reihenfolge, Format und Länge der im `CALLNAT`- bzw. `PERFORM`-Statement des aufrufenden Objekts angegebenen Parameter müssen genau mit Reihenfolge, Format und Länge der Felder, die im `DEFINE DATA PARAMETER`-Statement des aufgerufenen Subprogramms bzw. der aufgerufenen Subroutine definiert sind, übereinstimmen.

Die Namen der Variablen im aufrufenden Objekt und dem aufgerufenen Subprogramm bzw. der aufgerufenen Subroutine brauchen nicht dieselben zu sein (da die Übergabe der Parameter nach Speicheradressen erfolgt und nicht nach Namen).

Um zu garantieren, dass die im aufrufenden Programm benutzten Datenelement-Definitionen mit den im Subprogramm oder der externen Subroutine benutzten Datenelement-Definitionen identisch sind, können Sie eine PDA in einem `DEFINE DATA LOCAL USING`-Statement angeben. Wenn Sie eine PDA als eine LDA benutzen, können Sie sich den zusätzlichen Aufwand der Erstellung einer LDA ersparen, die dieselbe Struktur wie die PDA hat.