Bundesanstalt für Wasserbau – Außenstelle Küste

BAW-Datenformat BDF

Fortran-Schnittstelle
Technischer Bericht — Version 2.0

Bundesanstalt für Wasserbau

Außenstelle Küste

Wedeler Landstraße 157

22559 Hamburg-Rissen

Tel. 040 81908 0

Fax. 040 81908 373

http://www.hamburg.baw.de/

30. Dezember 1998

Inhaltsverzeichnis

1 Zielsetzung

Bei der BAW-AK werden unterschiedliche hydronumerische Verfahren für die Simulation von Strömungs- und Transportvorgängen eingesetzt. Derzeit handelt es sich hierbei um die Verfahren ARTEMIS, TELEMAC-2D, TRIM-2D, TRIM-3D, PARTRACE und WARM. Jedes Verfahren arbeitet mit seinem eigenen Datenformat, welches in der Regel eng mit der Struktur des jeweiligen Berechnungsgitters verknüpft ist. Die durchgängige Nutzung einheitlicher Datenformate wird dadurch verhindert, so daß zumindest für Preprocessing und Processing verfahrensspezifische Datenformate unterstützt werden müssen (z. B. für Topographie und Randwerte).

Hingegen bietet sich der Bereich des Postprocessing für den Einsatz einer einheitlichen *verfahrensunabhängigen* Daten- und Dateistruktur an. Dies ist deshalb wünschenswert, weil die Berechnungsergebnisse verschiedener mathematischer Modellverfahren in gleicher Weise analysiert und graphisch dargestellt werden müssen. Bei der BAW-AK wurde zu diesem Zwecke das sogenannte BAW-Datenformat (BDF) kreiert. Alle Berechnungsergebnisse der Modellverfahren werden in der Regel nach BDF konvertiert.

In diesem technischen Bericht wird das BDF-Datenformat ausführlich vorgestellt. Dies soll dem Leser die Verwendung von im BDF-Format abgelegten Daten oder das Schreiben neuer Dateien aus eigenen Anwendungsprogrammen heraus, unter Nutzung schon vorhandener Softwarebausteine, erleichtern bzw. ermöglichen.

Bei dem vorliegenden technischen Bericht handelt es sich um die erste Überarbeitung der Ursprungsfassung (Version 1.0 vom Januar 1998). Abweichungen hiervon werden durch einen Revisionsbalken am Rande des Textes kenntlich gemacht.

2 Einleitung

Dieser Abschnitt vermittelt einen kurzen Überblick zu den Zielen, Konzepten sowie zur Entwicklungsgeschichte des BAW-Datenformates (BDF).

2.1 Grundlegende Ziele

Mit der Realisierung des BDF-Datenformats sollten folgende Ziele erreicht werden:

- Verfahrensunabhängige Speicherung der Daten.
- Speichern aller relevanten Informationen über die Daten.
- Speichern unterschiedlicher Datenarten, namentlich
  - skalare,
  - vektorielle und
  - tensorielle

  Größen.
- Speichern unterschiedlicher (FORTRAN-) Datentypen, nämlich
  - ganzzahlige,

  - reellwertige,
  - doppeltgenaue und
  - komplexe

  Zahlen.

- Speichern ein-, zwei- und dreidimensionaler vektorieller und tensorieller Größen.
- Speichern unterschiedlicher Arten von Datensätzen, insbesondere
  - Zeitserien,
  - synoptische Berechnungsergebnisse sowie
  - abgeleitete (Analyse-) Ergebnisse.
- Rascher wahlfreier Zugriff auf einzelne Datensätze.
- Plattformunabhängigkeit.
- Standardisierte, anwendungsübergreifende Bezeichnung der Namen und Einheiten physikalischer Größen.

2.2 Konzeption

Die in dem vorangehenden Abschnitt 2.1 aufgezählten Ziele konnten durch Realisierung des nachfolgenden Konzepts erreicht werden.

- Daten und Informationen über Daten werden in drei zusammengehörenden Dateien abgelegt:

  ***filename*:** In dieser binären Direktzugriffsdatei (Kürzel `DD`) sind sämtliche Datensätze (verschiedene physikalische Größen, Zeitpunkte, ...) in der Form einzelner Datenrecords abgelegt. D. h. für jede abgespeicherte physikalische Größe kann auf jeden abgelegten Zeitpunkt unter Kenntnis der Recordnummer direkt und somit schnell zugegriffen werden. Alle Records in dieser Datei weisen dieselbe Länge auf. Informationen über Art und Reihenfolge der abgelegten Daten sind allerdings nicht in dieser Datei, sondern in der Datei mit der Endung `.I` enthalten.

  ***filename*`.R`:** In dieser ASCII-Datei ist die Recordlänge der Datei mit der Endung `.I` abgelegt. Diese Datei (Kürzel `RD`) wird von einem Anwendungsprogramm vor dem Öffnen der Datei mit der Endung `.I` (automatisch) gelesen.

  ***filename*`.I`:** In dieser binären Direktzugriffsdatei (Kürzel `RI`) sind alle Informationen über Art und Reihenfolge der Daten sowie die Größe der Datenrecords der Datei `DD` abgespeichert. Diese Informationen sind für das Lesen oder Schreiben der Datenrecords notwendig. Des weiteren können diese Informationen von dem Anwendungsprogramm auch zur Steuerung des Programmablaufs vorteilhaft ausgewertet werden.

  Die getrennte Speicherung der Daten von den Informationen über die Daten ermöglicht eine platzsparende Speicherung.

- Definition und Charakterisierung der (erlaubten) physikalischen Größen erfolgt in folgenden Konfigurationsdateien:

  `phydef.cfg.de.dat`: Diese Datei enthält die deutschen Bezeichnungen für alle gültigen physikalischen Größen zusammen mit ihren physikalischen Einheiten.

  `phydef.cfg.en.dat`: Diese Datei enthält die entsprechenden englischen Bezeichnungen der physikalischen Größen und Einheiten.

  `phydef.cfg.rest.dat`: In dieser Datei sind weitergehende, sprachunabhängige Charakterisierungen der physikalischen Größen festgelegt (z. B. ob es sich um eine skalare, vektorielle, ... Größe handelt).

  Alle vorgenannten Konfigurationsdateien befinden sich in dem Public-Verzeichnis `cfg` auf dem zentralen Server der BAW-AK.

Weitere Details zu Implementierung und Anwendung werden in den nachfolgenden Kapiteln vorgestellt.

2.3 Entwicklungsgeschichte

Die Arbeiten an einem Vorläufer des BDF-Formats wurden im Juli 1991 begonnen. Die wesentlichen Beschränkungen waren:

1. Datendateien und ihr Inhalt waren statisch vorkonfiguriert. Zu jedem *Dateityp* gehörte ein bestimmter *Inhalt* (so waren beispielsweise Reihenfolge und Art der physikalischen Größen anhand des Dateityps vorgegeben).

2. Neben Skalaren konnten nur zweidimensionale vektorielle Größen abgespeichert werden.

Diese Beschränkungen wurden im Rahmen einer im Dezember 1994 beginnenden grundlegenden Überarbeitung überwunden, die im Laufe des Jahres 1995 abgeschlossen werden konnte. Seither erfüllt BDF die in Abschnitt 2.1 geschilderten grundlegenden Anforderungen. Die Implementierung arbeitet stabil und (bislang) fehlerfrei.

Im Jahre 1998 wurden verschiedene Fortran90-Module, vor allem zum Lesen und Schreiben von Daten im BDF-Format entwickelt. Hierbei wurden insbesondere die Möglichkeiten zur dynamischen Dimensionierung von Feldern sowie zur Deklaration selbstdefinierter Datentypen intensiv genutzt. Dadurch war es möglich, das Benutzen des BDF-Formates in neu erstellten Anwendungsprogrammen erheblich zu erleichtern und vor allem auch wesentlich sicherer zu machen. Letzteres kommt insbesondere bei der nunmehr, gegenüber früher, stark vereinfachten Deklaration der Variablen und Felder zum Tragen.

3 Physikalische Größen

3.1 Speicherung der Definitionen physikalischer Größen

Die in dem nachfolgenden Abschnitt 3.2 angegebenen Größen werden als globale Daten in mit `save`-Attribut deklarierten Feldern und Variablen in dem (sogenannten) Includemodul

phycod_f90 vorgehalten. Auf diesem Wege wird, nach dem Lesen und Speichern, die Information zwischen den FORTRAN-Unterprogrammen verschiedener Bibliotheken und Anwendungsprogramme ausgetauscht. Die nachfolgend angebenen Variablennamen entsprechen den in dem Includemodul festgeschriebenen Namen.

3.1.1 Includemodul phycod_f90

Dateiinhalt: mit save-Attribut versehene Felder und Variablen, die zur Speicherung der in dem Abschnitt 3.2 detaillierter vorgestellten Informationen über physikalische Größen verwendet werden.

Beispiel: Datei phycod_f90.f90.

Directory: $(PROGHOME)/public/inc/<*system*>/.

Dimensionierung:

1. Maximale Anzahl MAXPHY der definierbaren physikalischen Größen.
2. Maximale Anzahl MAXPKLA der Klassen, in welche die physikalischen Größen weiter unterteilt werden können.

3.2 Definition physikalischer Größen

Die zu einer physikalischen Größe gehörenden beschreibenden Daten können nur dann in Anwendungsprogrammen verwendet werden, wenn diese zuvor in verschiedenen Konfigurationsdateien (siehe Abschnitt 2.2) definiert wurden. Nachfolgend erfahren Sie, welche Informationen in diesen Dateien abgelegt sind.

3.2.1 Datei phydef.cfg.de.dat

Es handelt sich um eine Datei im ASCII-Format. Die Datei kann wie folgt näher charakterisiert werden:

Beispiel: Datei phydef.cfg.de.dat.

Directory: $(PROGHOME)/public/cfg/

Dateiinhalt: Deutsche Bezeichnungen physikalischer Größen.

① Ganzzahlige Code-Nummer (1 ... MAXPHY).

② Name.
typ = CHARACTER,
var = CPHYS(MAXPHY)*40.

③ Einheit.
typ = CHARACTER,
var = CPHYSU(MAXPHY)*10.

Speicherung: In Includemodul phycod_f90 (siehe Abschnitt 3.1).

3.2.2 Datei `phydef.cfg.en.dat`

Es handelt sich um eine Datei im ASCII-Format. Die Datei kann wie folgt näher charakterisiert werden:

Beispiel: Datei `phydef.cfg.en.dat`.

Directory: `$(PROGHOME)/public/cfg/`

Dateiinhalt: Englische Bezeichnungen physikalischer Größen.

① Ganzzahlige Code-Nummer (1 ... `MAXPHY`).

② Name.
`typ = CHARACTER`,
`var = CPHYS(MAXPHY)*40`.

③ Einheit.
`typ = CHARACTER`,
`var = CPHYSU(MAXPHY)*10`.

Speicherung: In Includemodul `phycod_f90` (siehe Abschnitt 3.1).

3.2.3 Datei `phydef.cfg.rest.dat`

Es handelt sich um eine Datei im ASCII-Format. Die Datei kann wie folgt näher beschrieben werden:

Beispiel: Datei `phydef.cfg.rest.dat`.

Directory: `$(PROGHOME)/public/cfg/`.

Dateiinhalt: weitergehende Charakterisierung physikalischer Größen.

① Ganzzahlige Code-Nummer (1 ... `MAXPHY`).

② Art.
`typ = INTEGER`,
`var = IPHYSTYP(MAXPHY)`.

1 = `Skalar`.
2 = `Vektor`.
3 = `Tensor`.
–1 = nicht definiert.

③ Zeitabhängigkeit.
`typ = INTEGER`,
`var = IPHYSZTA(MAXPHY)`.

0 = nicht »zeitartig« (z. B. Anzahl).
1 = synoptisch (ein Zeitpunkt oder Ereignis).
2 = synthetisch (z. B. Mittelwert fuer Zeitraum).
–1 = nicht definiert.

④ Klassenzuordnung.
`typ = INTEGER`,
`var = IPHYSKLAS(MAXPHY)`.
Insgesamt stehen derzeit 16 verschiedene Klassen zur Wahl (z. B. Flut-, Ebbe-, Flutstrom- und Ebbestromgrößen).

⑤ Kurzbezeichnung.
`typ = CHARACTER`,
`var = CPHYSABK(MAXPHY)*8`.
Hierbei handelt es sich um ein (weitestgehend) klingendes Akronym für die physikalische Größe. Diese werden beispielsweise von dem Programm HVIEW2D in den Textfeldern der Auswahlmenüs verwendet.

⑥ Zuordnung der Referenzwasserfläche.
`typ = INTEGER`,
`var = IPHYSREF(MAXPHY)`.
Die Referenzwasserfläche wird in Analyse- oder Graphikprogrammen z. B. dazu benötigt, um bei Kenntnis der Topographie des Modellgebietes trockengefallene Flächen von mit Wasser überfluteten Gebieten unterscheiden zu können. Durch Verschneiden dieser (realen oder künstlichen) »Wasseroberfläche« mit der Topographie wird gleichsam das Gebiet festgelegt, in dem definierte Daten vorliegen.

N = die physikalische Größe mit der Code-Nummer *N* ist die Referenzwasserfläche (z. B. ist der Wasserstand die Referenzwasserfläche des Salzgehalts).

0 = keine Referenzwasserfläche notwendig (z. B. für Anzahl und Luftdruck).

–1 = undefiniert.

⑦ Multiplikator.
`typ = REAL`,
`var = RPHYMUL(MAXPHY)`.
In den Datendateien des Typs `DD` (siehe Abschnitt 2.2) sind die physikalischen Größen in der Regel in Einheiten des MKSA-Systems abgspeichert. Zur graphischen Darstellung ist es oftmals zweckmäßig, diese auf eine praktikablere Einheit umzurechnen (z. B. in die in `phydef.cfg.de.dat` angegebene).

⑧ Beschriftungsindex.
`typ = INTEGER`,
`var = IPHYMZR(MAXPHY)`.
Graphikprogramme müssen oftmals eine zu der dargestellten Größe passende »Zeitangabe« in der Zeichnung anbringen.

0 = Ohne Beschriftung.

1 = Zeitangabe oder Tideuhr für einen synoptischen Zeitpunkt.

2 = Zeitraum von . . . bis . . . .

3 = Tideanfang.

4 = Tidehochwasserzeit.

5 = Halbtidebeginn.

6 = Kenterungszeit.

7 = Tideniedrigwasserzeit.

Speicherung: In Includemodul `phycod_f90` (siehe Abschnitt 3.1).

4 Info-Records der Daten

Um mit im BDF-Format vorliegenden Daten arbeiten zu können, halten Anwendungsprogramme die in den BDF-Dateien des Typs `RI` (siehe Abschnitt 2.2) abgelegten Daten üblicherweise in diversen COMMON-Blöcken vor. Diese COMMON-Blöcke sind Teil der Include-Datei `direkt.h`. Über sie läuft im wesentlichen der Informationsaustausch zwischen grundlegenden FORTRAN-Unterprogrammen verschiedener Bibliotheken bei vielen Anwendungen.

Das Auslesen und Eintragen der aktuellen Werte aus bzw. in die Variablen und Felder der Include-Datei `direkt.h` geschieht meist indirekt über einen Satz von Unterprogrammschnittstellen – zu jedem Info-Record-Typ gehört genau ein Lese- und ein Schreibunterprogramm (siehe hierzu den nachfolgenden Abschnitt 4.2) – oder über einige wenige High-Level Schnittstellenunterprogramme wie `dadrir.f` und `dadgir.f` (unter FORTRAN77) sowie `BdfInfoRead` und `BdfInfoWrite` (in dem Fortran90-Modul `mod_bdf_info_io`). Das Verwenden dieser höherwertigen Schnittstellen wird zu einem späteren Zeitpunkt beschrieben.

4.1 Speicherung der Info-Records der Daten

Die in dem nachfolgenden Abschnitt 4.2 angegebenen Maximalgrößen sind namensgleich mit den zur Deklaration verschiedener Variablen und Felder benutzten maximalen Feldgrenzen, die in verschiedenen benannten COMMON-Blöcken der Include-Datei `direkt.h` abgelegt sind.

4.1.1 Include-Datei `direkt.h`

Dateiinhalt: COMMON-Blöcke mit allen Feld- und Variablennamen, die zur Speicherung der in dem Abschnitt 4.2 detaillierter vorgestellten Informationen über Daten verwendet werden.

Beispiel: Datei `direkt.h`.

Directory: `$(PROGHOME)/public/inc/`.

Dimensionierung:

1. Maximale Anzahl `MDFILE` der BDF-Dateien des Typs `DD`, für die gleichzeitig in einem Anwendungsprogramm Informationen über die darin abgelegten Daten vorgehalten werden können.
2. Zu jeder BDF-Datei können für maximal `MIFILE` weitere sogenannte INP-Dateien (z. B. Systemdatei, der Datei mit Geometrie des Modellsystems) zusätzliche Kurzinformationen abgelegt sein (INP-Dateibeschreibung).
3. Jede INP-Datei kann durch maximal `MIFII` Attribute näher beschrieben werden.
4. In jeder BDF-Datei des Typs `RI` können mehrere Datumsangaben (charakteristischer Zeitpunkt) enthalten sein, die jeweils mit maximal `MZEI1` Angaben näher beschrieben werden können.
5. In jeder BDF-Datei des Typs `RI` können maximal `MZEI2` verschiedene Datumsangaben der Art >charakteristischer Zeitpunkt< enthalten sein.

6. Für bis zu maximal MPINF verschiedene physikalische Größen können in einer BDF-Datei des Typs RI beschreibende Informationen abgespeichert sein.
7. Sollten in einer BDF-Datei des Typs DD äquidistante Zeitserien abgespeichert sein, so kann diese Zeitreihe in der dazugehörigen BDF-Datei des Typs RI durch maximal MADZRI Angaben näher beschrieben werden (Charakterisierung Zeitreihe).
8. Maximale Anzahl MAXCED zulässiger unterschiedlicher Dateitypen.
9. Maximale Anzahl MAXFOR zulässiger unterschiedlicher FORTRAN-Dateiformate.
10. Maximale Anzahl MAXACC zulässiger unterschiedlicher FORTRAN-Dateizugriffsarten.
11. Maximale Anzahl MAXBEZ zulässiger unterschiedlicher charakteristischer Zeitpunkte.
12. In einer BDF-Datei des Typs RI können bis zu maximal MAXCDI Kommentare enthalten sein.
13. Bis zu maximal MINFREC unterschiedliche Info-Record-Typen können in einer BDF-Datei des Typs RI in beliebiger Reihenfolge enthalten sein.
14. Bis zu maximal MRCREC Info-Records können in einer BDF-Datei des Typs RI abgespeichert sein.
15. Bis zu maximal MOZ Informationen über verschiedene Ausgabezeitpunkte (z. B. Datumsangaben für synoptische Ergebniszustände) können in einer BDF-Datei des Typs RI abgespeichert sein.
16. Jeder Ausgabezeitpunkt kann durch bis zu maximal MOUZEI Attribute näher beschrieben werden.
17. Zu jeder BDF-Datei können für maximal MIOFI weitere sogenannte OUT-Dateien (z. B. die BDF-Datei des Typs DD) zusätzliche Kurzinformationen abgelegt sein (OUT-Dateibeschreibung).
18. Jede OUT-Datei kann durch maximal MIOINF Attribute näher beschrieben werden.
19. Maximale Anzahl MAXPOS zulässiger unterschiedlicher Positionsbezeichnungen.
20. Jede Positionsbezeichnung kann durch bis zu maximal MIPOS ganzzahlige Attribute näher beschrieben werden.
21. Jede Positionsbezeichnung kann durch bis zu maximal MRPOS reellwertige Attribute näher beschrieben werden.
22. Maximale Anzahl MPOS in einer Datei des Typs RI abgelegter Positionsbezeichnungen.
23. Maximale Anzahl MAXPTYP zulässiger unterschiedlicher Arten physikalischer Größen.
24. Maximale Anzahl MAXNORM der in einer BDF-Datei des Typs RI zu einer physikalischen Größe abgelegten Informationen über die Normierung der in der BDF-Datei des Typs DD abgespeicherten Daten.
25. Maximale Anzahl MAXREPL der in einer BDF-Datei des Typs RI zu einer physikalischen Größe abgelegten Informationen über die Datenersetzung für die in der BDF-Datei des Typs DD abgespeicherten Daten.

26. Maximale Anzahl `MAXINFO` der in einer BDF-Datei des Typs `RI` abgelegten Zusatzinformationen (z. B. Grenzwassertiefe für das Trockenfallen und Überfluten von Wattgebieten).
27. Maximale Anzahl `MAXHZZZ` der in einer BDF-Datei des Typs `RI` abgelegten Angaben zu Tiefenschichten, mit denen die Vertikalstruktur einer dreidimensionalen Modellgeometrie näher beschrieben werden kann.

4.2 Definition der Info-Records der Daten

Die in Abschnitt 4.2 angegebenen Größen sind i. d. R. *nicht* namensgleich mit den in verschiedenen benannten COMMON-Blöcken der Include-Datei `direkt.h` abgelegten Feldern und Variablen. In den meisten mit dem BDF-Format arbeitenden Anwendungsprogrammen erfolgt das Auslesen und Eintragen der aktuellen Werte, aus bzw. in die Variablen und Felder der Include-Datei `direkt.h`, meist mit Hilfe der High-Level Unterprogrammschnittstellen `dadrir.f` und `dadgir.f` (beide FORTRAN77-Unterprogramme im Public-Verzeichnis `record`) oder `BdfInfoRead` und `BdfInfoWrite` (falls die in dem Fortran90-Modul `mod_bdf_info_io` vorhandenen Unterprogramme verwendet werden). Die nachfolgend benutzten Namen entsprechen entweder (FORTRAN77) den in diesen Unterprogrammen benutzten Variablen- und Feldnamen oder aber (Fortran90) den Komponentennamen der selbstdefinierten Datentypen.

Die Informationen über abgespeicherte Daten sind in der Form einzelner Records in der BDF-Datei des Typs `RI` abgespeichert. Es sind maximal `MINFREC` unterschiedliche Info-Recordtypen definiert. Für jeden Typ gib es auf der untersten Anwendungsebene jeweils ein FORTRAN-Unterprogramm für das Lesen und Schreiben der Daten aus bzw. in die Variablen und Felder der Include-Datei `direkt.h`.

Nachfolgend werden für jeden gültigen Info-Recordtyp die Bedeutung und Definition der relevanten Größen angegeben. Ferner werden die für Lesen und Schreiben vorhandenen Low-Level FORTRAN-Unterprogramme aufgeführt.

4.2.1 Info-Recordtyp 000

Lesen und Schreiben einiger elementarer Größen:

Auslesen: FORTRAN-Unterprogramm `get000.f`

Speichern: FORTRAN-Unterprogramm `rec000.f`

Directory: `$(PROGHOME)/public/record/`<*system*>`/`

Record-Status: Record muß vorhanden sein.

Record-Inhalt: Informationen für den Dateikopf der BDF-Datei des Typs `RI`:

① Nummer aktuelle Datei, 1 . . . `MDFILE`.
`typ = INTEGER`,
`var = LAKTDF`.
Mit Hilfe dieser Zahl wird ein *Pointer* gesetzt, der auf die Informationen der `LAKTDF`-ten BDF-Datei in den Feldern der Include-Datei `direkt.h` verweist.

② Code-Dateityp, 1 ... MAXCED.
typ = INTEGER,
var = LDFTYP.
Mit dieser Zahl wird ein *Pointer* auf eine Position in dem Feld CIDCED in der Include-Datei direkt.h erzeugt.

- Liegen die Werte im Intervall 1 ... 120, so handelt es sich um einen (historisch) festgelegten Dateityp, d. h. der Inhalt ist durch den Typ schon eindeutig festgelegt. Wichtige Beispiele sind
  - LDFTYP=1, dann handelt es sich um eine Systemdatei des Typs sysdat, oder es ist
  - LDFTYP=121, dann ist die Systemdatei vom Typ gitter05, beziehungsweise falls
  - LDFTYP=123 ist, dann handelt es sich um eine Systemdatei des Typs profil05.
- Liegen die Werte hingegen im Intervall 121 ... 130, dann handelt es sich um *dynamisch* konfigurierte Dateien, die keinem bestimmten Typ zugeordnet werden können.

③ Kürzel-Dateityp.
typ = CHARACTER,
var = LCIDCED*10.
In dem Unterprogramm dirdef1.f (Public-Verzeichnis record) wird die Zuordnung zwischen Kürzel-Dateityp und Code-Dateityp festgelegt.

4.2.2 Info-Recordtyp 001

Lesen und Schreiben der näheren Charakterisierung einer INP-Datei. Jede BDF-Datei des Typs RI muß an erster Stelle die Beschreibung der zu den in der Datei des Typs DD abgespeicherten Daten gehörenden Systemdatei enthalten (siehe auch Abschnitt 4.2.1 zur Bedeutung der dortigen Variablen LDFTYP). Des weiteren können Angaben zu Dateien enthalten sein, aus denen z. B. die in der BDF-Datei des Typs DD abgelegten Daten hervorgegangen sind. So erzeugt das Programm ZEITR aus synoptischen Datensätzen Zeitserien für alle Punkte des Berechnungsgitters. In den Angaben zu den INP-Dateien kann daher auch die Verarbeitungsgeschichte der Daten festgehalten werden.

Auslesen: FORTRAN-Unterprogramm get001.f; es werden mit dem UP-Aufruf alle Charakterisierungen ausgelesen.

Speichern: FORTRAN-Unterprogramm rec001.f; es wird mit dem UP-Aufruf nur eine Charakterisierung geschrieben.

Directory: $(PROGHOME)/public/record/<*system*>/

Record-Status: Record muß vorhanden sein. Die Beschreibung der zu den Daten gehörenden Systemdatei muß enthalten sein.

Record-Inhalt: Dateiname und Attribute für eine INP-Dateibeschreibung:

① Anzahl INP-Dateien.
typ = INTEGER,
var = NIF.
Diese Zahl wird nur beim Auslesen ermittelt und liefert die Anzahl der in der ausgewählten BDF-Datei des Typs RI beschriebenen INP-Dateien.

② Code INP-Dateiformat.
typ = INTEGER,
var = IFFORM(MAXNIF), mit MAXNIF $\geq$ NIF.
Mit dieser Zahl wird ein *Pointer* auf eine Position in dem Feld CIDFOR in der Include-Datei direkt.h erzeugt.

1 = UNFORMATTED.

2 = FORMATTED.

③ Code INP-Dateizugriff.
typ = INTEGER,
var = IFACC(MAXNIF), mit MAXNIF $\geq$ NIF.
Mit dieser Zahl wird ein *Pointer* auf eine Position in dem Feld CIDACC in der Include-Datei direkt.h erzeugt.

1 = SEQUENTIAL.

2 = DIRECT.

④ Code INP-Dateityp, 1 ... MAXCED.
typ = INTEGER,
var = IFCIDCED(MAXNIF), mit MAXNIF $\geq$ NIF.
Mit dieser Zahl wird ein *Pointer* auf eine Position in dem Feld CIDCED in der Include-Datei direkt.h erzeugt. Siehe hierzu die zu der Variablen LDFTYP in dem Abschnitt 4.2.1 gegebenen Erläuterungen.

⑤ INP-Dateiname, ohne Pfadangabe.
typ = CHARACTER,
var = CIF(MAXNIF)*80, mit MAXNIF $\geq$ NIF.
Die Namenslänge der INP-Datei darf 38 Zeichen nicht übersteigen, da ggf. die feststehenden Dateiendungen .R und .I ergänzt werden (siehe Abschnitt 2.2), und in der Include-Datei direkt.h Dateinamen nur eine maximale Länge von 40 Zeichen aufweisen dürfen.

4.2.3 Info-Recordtyp 002

Lesen und Schreiben der näheren Beschreibung charakteristischer Zeitpunkte. Es können ggf. mehrere (sinnvolle) Angaben in einer BDF-Datei des Typs RI enthalten sein. Beispielsweise greifen die Graphikprogramme HVIEW2D, VVIEW2D und LQ2PRO auf diese Angaben zurück.

Auslesen: FORTRAN-Unterprogramm get002.f; es werden mit dem UP-Aufruf alle Charakterisierungen ausgelesen.

Speichern: FORTRAN-Unterprogramm rec002.f; es wird mit dem UP-Aufruf nur eine Charakterisierung geschrieben.

Directory: `$(PROGHOME)/public/record/`<*system*>`/`

Record-Status: Record kann vorhanden sein. Die Beschreibung eines Bezugszeitpunktes sollte vorhanden sein.

Record-Inhalt: Beschreibung eines charakteristischen Zeitpunkts:

① Anzahl charakt. Zeitpunkte.
`typ = INTEGER`,
`var = NZP`.
Diese Zahl wird nur beim Auslesen ermittelt und liefert die Anzahl der in der ausgewählten BDF-Datei des Typs `RI` beschriebenen charakteristischen Zeitpunkte.

② Code charakt. Zeitpunkt.
`typ = INTEGER`,
`var = BCIDBEZ(MAXNZP)`, mit `MAXNZP` $\geq$ `NZP`.
Mit dieser Zahl wird ein *Pointer* auf eine Position in dem Feld `CIDBEZ` in der Include-Datei `direkt.h` erzeugt.

1 = `Bezugszeitpunkt` (= Zeit-Ursprung)

2 = `Anfangszeitpunkt`

3 = `Endzeitpunkt`

Anfangszeitpunkt und Endzeitpunkt werden beispielsweise von den zur Datenanalyse verwendeten Programmen TDKWF, TDKVF, TDKSF, TDKLF, LZKWF und LZKSF benutzt, um Anfang und Ende des Analysezeitraums festzuhalten.

③ Jahr.
`typ = INTEGER`,
`var = BJAHR(MAXNZP)`, mit `MAXNZP` $\geq$ `NZP`.

④ Monat.
`typ = INTEGER`,
`var = BMONAT(MAXNZP)`, mit `MAXNZP` $\geq$ `NZP`.

⑤ Tag.
`typ = INTEGER`,
`var = BTAG(MAXNZP)`, mit `MAXNZP` $\geq$ `NZP`.

⑥ Stunde.
`typ = INTEGER`,
`var = BSTUNDE(MAXNZP)`, mit `MAXNZP` $\geq$ `NZP`.

⑦ Minute.
`typ = INTEGER`,
`var = BMINUTE(MAXNZP)`, mit `MAXNZP` $\geq$ `NZP`.

⑧ Sekunde.
`typ = INTEGER`,
`var = BSEKUNDE(MAXNZP)`, mit `MAXNZP` $\geq$ `NZP`.

⑨ Nanosekunde.
`typ = INTEGER`,
`var = BNANOSEKUNDE(MAXNZP)`, mit `MAXNZP` $\geq$ `NZP`.

4.2.4 Info-Recordtyp 003

Lesen und Schreiben der Anzahl der Datenpunkte des zumeist zwei- oder dreidimensionalen Datenraums, für die in einer BDF-Datei des Typs `DD` Daten-Records abgespeichert sind.

Auslesen: FORTRAN-Unterprogramm `get003.f`.

Speichern: FORTRAN-Unterprogramm `rec003.f`.

Directory: `$(PROGHOME)/public/record/`<*system*>`/`

Record-Status: Record muß vorhanden sein.

Record-Inhalt: Anzahl der Datenpunkte:

① Anzahl der Datenpunkte.
`typ = INTEGER`,
`var = IAKTKNO`.

4.2.5 Info-Recordtyp 004

Lesen und Schreiben der Liste der physik. Größen, für die in einer BDF-Datei des Typs `DD` Daten-Records abgespeichert sind. Mit diesen in der BDF-Datei des Typs `RI` abgelegten Informationen ist es möglich, die physikalische Bedeutung der abgespeicherten Daten zu beschreiben.

Auslesen: FORTRAN-Unterprogramm `get004.f`.

Speichern: FORTRAN-Unterprogramm `rec004.f`.

Directory: `$(PROGHOME)/public/record/`<*system*>`/`

Record-Status: Record muß vorhanden sein.

Record-Inhalt: Anzahl und Code-Nummer(n) der abgespeicherten physikalischen Größen:

① Anzahl physik. Größen.
`typ = INTEGER`,
`var = IAKTPG`.

② Code physik. Größen.
`typ = INTEGER`,
`var = LISTEPG(MAXPG)`, mit `MAXPG` $\geq$ `IAKTPG`.
Die Code-Nummer muß dabei den in den Konfigurationsdateien festgelegten Werten entsprechen (siehe Abschnitt 3.2).

4.2.6 Info-Recordtyp 005

Lesen und Schreiben von Kommentaren. Es können ggf. mehrere (sinnvolle) Kommentarzeilen in einer BDF-Datei des Typs `RI` abgelegt werden. Diese Zeilen dienen dem besseren Verständnis der in der BDF-Datei des Typs `DD` abgespeicherten Daten. Anwendungsprogramme, so z. B. das Programm VTDK, führen selbständig Ergänzungen zu den eventuell schon vorhandenen Kommentaren durch.

Auslesen: FORTRAN-Unterprogramm `get005.f`; es werden mit dem UP-Aufruf alle Kommentare ausgelesen.

Speichern: FORTRAN-Unterprogramm `rec005.f`; es wird mit dem UP-Aufruf nur ein Kommentar gespeichert.

Directory: `$(PROGHOME)/public/record/`<*system*>`/`

Record-Status: Record kann vorhanden sein.

Record-Inhalt: Klartext-Kommentarzeilen:

① Anzahl Kommentare.
`typ = INTEGER`,
`var = NCO`.
Diese Zahl wird nur beim Auslesen ermittelt und liefert die Anzahl der in der ausgewählten BDF-Datei des Typs `RI` vorhandenen Kommentare.

② Klartext Kommentarzeile.
`typ = CHARACTER`,
`var = CO(MAXCO)*80`, mit `MAXCO` $\geq$ `NCO`.
Jede Kommentarzeile darf beliebige ASCII-Zeichen enthalten.

4.2.7 Info-Recordtyp 006

Handelt es sich bei den in der BDF-Datei des Typs `DD` abgespeicherten Daten um Zeitserien mit äquidistanten Stützstellenabständen, so muß in der BDF-Datei des Typs `RI` diese näher beschrieben werden (Charakterisierung Zeitreihe).

Auslesen: FORTRAN-Unterprogramm `get006.f`.

Speichern: FORTRAN-Unterprogramm `rec006.f`.

Directory: `$(PROGHOME)/public/record/`<*system*>`/`

Record-Status: Record muß dann vorhanden sein, wenn es sich bei den in der BDF-Datei des Typs `DD` abgespeicherten Daten um eine äquidistante Zeitserie handelt.

Record-Inhalt: nähere Charakterisierung der Zeitreihe:

① Anfangszeit Sekunden.
`typ = INTEGER`,
`var = ANFANGSZEIT`.
Diese Zeitangabe bezieht sich auf den Zeit-Ursprung, und bezeichnet die seither verstrichenen Sekunden für den Beginn der Zeitreihe. Der Zeit-Ursprung wird mit Hilfe des Info-Records >charakteristischer Zeitpunkt< als >Bezugszeitpunkt< festgelegt (siehe auch Abschnitt 4.2.3).

② Anfangszeit Nanosekunden.
`typ = INTEGER`,
`var = NANOANFANG`.
Sollte der Beginn der Zeitreihe in Bezug auf den Zeit-Ursprung (siehe oben) so gelegen sein, daß in der Zeitdifferenz Sekundenbruchteile enthalten sind, so gibt `NANOANFANG` diesen Sekunden-Bruchteil in Nanosekunden an.

③ Zeitschritt Sekunden.
typ = INTEGER,
var = ZEITSCHRITT.
Diese Angabe bezeichnet den Sekunden-Anteil am äquidistanten Zeitschritt zwischen zwei aufeinanderfolgenden Stützstellen der Zeitreihe.

④ Zeitschritt Nanosekunden.
typ = INTEGER,
var = NANOSCHRITT.
Diese Angabe bezeichnet den Sekunden-Bruchteil am äquidistanten Zeitschritt zwischen zwei aufeinanderfolgenden Stützstellen der Zeitreihe, angegeben in Nanosekunden.

⑤ Anzahl Stützstellen.
typ = INTEGER,
var = ANZAHL.
Die Anzahl der Intervalle der Zeitreihe ist um den Wert 1 kleiner als die Stützstellenzahl ANZAHL.

4.2.8 Info-Recordtyp 007

Lesen und Schreiben der näheren Charakterisierung einer OUT-Datei. Jede BDF-Datei des Typs RI muß mindestens die Beschreibung der die Daten enthaltenden Datei des Typs DD enthalten; ohne diese Informationen kann diese Datei später nicht geöffnet werden.

Auslesen: FORTRAN-Unterprogramm get007.f; es werden mit dem UP-Aufruf alle Charakterisierungen ausgelesen.

Speichern: FORTRAN-Unterprogramm rec007.f; es wird mit dem UP-Aufruf nur eine Charakterisierung geschrieben.

Directory: $(PROGHOME)/public/record/<*system*>/

Record-Status: Record muß zur Beschreibung der zugehörigen BDF-Datei des Typs DD vorhanden sein.

Record-Inhalt: Dateiname und Attribute für eine OUT-Dateibeschreibung:

① Anzahl OUT-Dateien.
typ = INTEGER,
var = NOF.
Diese Zahl wird nur beim Auslesen ermittelt und liefert die Anzahl der in der ausgewählten BDF-Datei des Typs RI beschriebenen OUT-Dateien.

② Code OUT-Dateiformat.
typ = INTEGER,
var = OFFORM(MAXNOF), mit MAXNOF $\geq$ NOF.
Mit dieser Zahl wird ein *Pointer* auf eine Position in dem Feld CIDFOR in der Include-Datei direkt.h erzeugt.

1 = UNFORMATTED.

2 = FORMATTED.

③ Code OUT-Dateizugriff.
typ = INTEGER,
var = OFACC(MAXNOF), mit MAXNOF $\geq$ NOF.
Mit dieser Zahl wird ein *Pointer* auf eine Position in dem Feld CIDACC in der Include-Datei direkt.h erzeugt.

1 = SEQUENTIAL.

2 = DIRECT.

④ Code OUT-Dateityp, 1 ... MAXCED.
typ = INTEGER,
var = OFCIDCED(MAXNOF), mit MAXNOF $\geq$ NOF.
Mit dieser Zahl wird ein *Pointer* auf eine Position in dem Feld CIDCED in der Include-Datei direkt.h erzeugt. Siehe hierzu die zu der Variablen LDFTYP in dem Abschnitt 4.2.1 gegebenen Erläuterungen.

⑤ OUT-Dateiname, ohne Pfadangabe.
typ = CHARACTER,
var = COF(MAXNOF)*(80), mit MAXNOF $\geq$ NOF.
Die Namenslänge der OUT-Datei darf 38 Zeichen nicht übersteigen, da ggf. die feststehenden Dateiendungen .R und .I ergänzt werden (siehe Abschnitt 2.2), und in der Include-Datei direkt.h Dateinamen nur eine maximale Länge von 40 Zeichen aufweisen dürfen.

⑥ Recordlänge OUT-Datei.
typ = INTEGER,
var = OFRECL(MAXNOF), mit MAXNOF $\geq$ NOF.
Um das Arbeiten mit Direktzugriffsdateien wie der BDF-Datei des Typs DD auf verschiedenen Rechnerplattformen zu ermöglichen wird OFRECL in Bytes angegeben. Erleichtert wird die korrekte Berechnung der Recordlänge durch das Unterprogramm dademrl.f; dieses Unterprogramm wird unter anderem von der High-Level-Schnittstelle dadgir.f gerufen (im Public-Verzeichnis record). Das korrekte Öffnen einer BDF-Datei des Typs DD leistet dadodf.f in Zusammenarbeit mit uniopfi.f.

Mit Ausnahme von ›Recordlänge OUT-Datei‹ entsprechen die o. g. Angaben denjenigen für INP-Dateien (siehe Abschnitt 4.2.2).

4.2.9 Info-Recordtyp 008

Lesen und Schreiben der näheren Beschreibung einer Positionsbezeichnung. Es können ggf. mehrere (sinnvolle) Angaben in einer BDF-Datei des Typs RI enthalten sein. Verschiedene Programme, wie z. B. VTDK, nutzen diese Positionsangaben.

Auslesen: FORTRAN-Unterprogramm get008.f; es werden mit dem UP-Aufruf alle Charakterisierungen ausgelesen.

Speichern: FORTRAN-Unterprogramm rec008.f; es wird mit dem UP-Aufruf nur eine Charakterisierung geschrieben.

Directory: $(PROGHOME)/public/record/<*system*>/

Record-Status: Record kann vorhanden sein.

Record-Inhalt: Beschreibung einer Positionsbezeichnung:

① Anzahl Positionsbezeichnungen.
`typ = INTEGER`,
`var = NBP`.
Diese Zahl wird nur beim Auslesen ermittelt und liefert die Anzahl der in der ausgewählten BDF-Datei des Typs `RI` beschriebenen Positionsbezeichnungen.

② Code Positionsbezeichnung.
`typ = INTEGER`,
`var = BPTYP(MAXNBP)`, mit `MAXNBP` $\geq$ `NBP`.
Mit dieser Zahl wird ein *Pointer* auf eine Position in dem Feld `CIDPOS` in der Include-Datei `direkt.h` erzeugt.

1 = `Referenzposition`
2 = `Bezugsposition`

Beide Arten von Positionen werden beispielsweise von den zur Datenanalyse verwendeten Programmen TDKWF, TDKVF, TDKSF und TDKLF benutzt, um entweder an diesen Positionen bestimmte Aktionen durchzuführen (z. B. Festlegung von Anzahl und Zeiten der Hoch- und Niedrigwasserereignisse innerhalb des Analysezeitraums an der Referenzposition) oder um Zeitdifferenzen (z. B. die Tidehochwasserzeit relativ zur Bezugsposition) zu bestimmen.

③ Knotennummer Position.
`typ = INTEGER`,
`var = BPNODE(MAXNBP)`, mit `MAXNBP` $\geq$ `NBP`.
Bezeichnung der Nummer (1 ... `IAKTKNO`) des Knotens in dem zumeist zwei- oder dreidimensionalen Datenraum, welcher der Lage der charakteristischen Position entspricht.

④ x-Koordinate Position (in m).
`typ = REAL`,
`var = XBP(MAXNBP)`, mit `MAXNBP` $\geq$ `NBP`.

⑤ y-Koordinate Position (in m).
`typ = REAL`,
`var = YBP(MAXNBP)`, mit `MAXNBP` $\geq$ `NBP`.

⑥ z-Koordinate Position (in m).
`typ = REAL`,
`var = ZBP(MAXNBP)`, mit `MAXNBP` $\geq$ `NBP`.

4.2.10 Info-Recordtyp 009

Lesen und Schreiben der Anzahl der zu einer Datengruppe gehörenden physikalischen Größen.

Auslesen: FORTRAN-Unterprogramm `get009.f`.

Speichern: FORTRAN-Unterprogramm `rec009.f`.

Directory: `$(PROGHOME)/public/record/`<*system*>`/`

Record-Status: Record muß vorhanden sein.

Record-Inhalt: Größe der Datengruppe:

① Größe Datengruppe.
`typ = INTEGER`,
`var = IAKTGRPG`, mit `IAKTGRPG` $\leq$ `IAKTPG`.
Sind in einer BDF-Datei `IAKTPG` physikalische Größen abgespeichert (siehe Abschnitt 4.2.5 unter ›Anzahl physik. Größen‹), so können für einen Teil davon mehrere, nämlich `NAZ` Datensätze (siehe Abschnitt 4.2.11 unter ›Anzahl Ausgabezeitpunkte), in der BDF-Datei des Typs `DD` vorkommen. Die zu diesem Teil gehörenden Datensätze müssen die ersten `IAKTGRPG` Größen in der Liste der abgespeicherten Daten `LISTEPG` sein.
Für Zeitserien und synoptische Datensätze sind `IAKTGRPG` und `IAKTPG` gleich groß, wohingegen sie bei Analysegrößen meist voneinander abweichen. So wird beispielsweise das Tidehochwasser für jede Tide innerhalb des Analysezeitraums ermittelt und damit auch mehrfach abgespeichert, während die für den Analysezeitraum berechneten Maximal-, Minimal- und Mittelwerte nur einmal in der BDF-Datei abgelegt werden. Letztere Größen gehören daher in diesem Fall nicht mehr zur Datengruppe.

4.2.11 Info-Recordtyp 010

Lesen und Schreiben der Ausgabezeitpunkte für Daten, bei denen es sich nicht um eine äquidistante Zeitreihe handelt (siehe hierzu Abschnitt 4.2.7). Bei den hier erforderlichen Zeitangaben handelt es sich im Falle synoptischer Datensätze jeweils um das Datum, für welches die Ergebnisse vorliegen. Bei abgeleiteten Größen (Analyseergebnissen) werden meistens die Zeiten abgespeichert, zu denen das Ereignis (z. B. das Tidehochwasser) an der Bezugsposition auftritt. Nur für die zu der `IAKTGRPG` physikalische Größen umfassenden Datengruppe (siehe Abschnitt 4.2.10 unter ›Größe Datengruppe‹) müssen Ausgabezeitpunkte angegeben werden. Die Ausgabezeitpunkte sind für alle `IAKTGRPG` Größen gleich.

Auf die zu den Ausgabezeitpunkten in der BDF-Datei des Typs `RI` abgelegten Angaben greifen u. a. die Graphikprogramme HVIEW2D, VVIEW2D und LQ2PRO zurück, um dem Benutzer die interaktive Auswahl des darzustellenden Ergebniszeitpunktes zu ermöglichen.

Auslesen: FORTRAN-Unterprogramm `get010.f`; es werden mit dem UP-Aufruf alle Ausgabezeitpunkte ausgelesen.

Speichern: FORTRAN-Unterprogramm `rec010.f`; es wird mit dem UP-Aufruf nur ein Ausgabezeitpunkt geschrieben.

Directory: `$(PROGHOME)/public/record/`<*system*>`/`

Record-Status: Record muß dann vorhanden sein, wenn z. B. entweder synoptische Datensätze oder tidebezogene Analyseergebnisse abgespeichert werden.

Record-Inhalt: Beschreibung eines Ausgabezeitpunkts:

① Anzahl Ausgabezeitpunkte.
typ = INTEGER,
var = NAZ.
Diese Zahl wird nur beim Auslesen ermittelt und liefert die Anzahl der in der ausgewählten BDF-Datei des Typs RI enthaltenen Ausgabezeitpunkte der zu der Datengruppe gehörenden Datensätze.

② Jahr.
typ = INTEGER,
var = AZJAHR(MAXNAZ), mit MAXNAZ $\geq$ NAZ.

③ Monat.
typ = INTEGER,
var = AZMONAT(MAXNAZ), mit MAXNAZ $\geq$ NAZ.

④ Tag.
typ = INTEGER,
var = AZTAG(MAXNAZ), mit MAXNAZ $\geq$ NAZ.

⑤ Stunde.
typ = INTEGER,
var = AZSTUNDE(MAXNAZ), mit MAXNAZ $\geq$ NAZ.

⑥ Minute.
typ = INTEGER,
var = AZMINUTE(MAXNAZ), mit MAXNAZ $\geq$ NAZ.

⑦ Sekunde.
typ = INTEGER,
var = AZSEKUNDE(MAXNAZ), mit MAXNAZ $\geq$ NAZ.

⑧ Nanosekunde.
typ = INTEGER,
var = AZNANOSEKUNDE(MAXNAZ), mit MAXNAZ $\geq$ NAZ.

4.2.12 Info-Recordtyp 011

Lesen und Schreiben der Differenzzeitpunkte für Daten. Bei den hier erforderlichen Angaben handelt es sich um zu den zuvor genannten Ausgabezeitpunkten gleichwertige Informationen (siehe Abschnitt 4.2.11). Diese Angaben werden in Ergänzung zu diesen von Programmen wie VTDK in die BDF-Datei des Typs RI geschrieben, wenn Differenzen zwischen den für zwei Untersuchungen A und B berechneten Analysegrößen gebildet werden sollen. Die BDF-Datei des Typs RI enthält dann sowohl die Zeitangaben für A wie auch für B, so daß auch zu einem späteren Zeitpunkt noch zweifelsfrei geklärt werden kann, aus welchen Einzelereignissen eine Differenzgröße berechnet wurde.

Auslesen: FORTRAN-Unterprogramm get011.f; es werden mit dem UP-Aufruf alle Differenzzeitpunkte ausgelesen.

Speichern: FORTRAN-Unterprogramm rec011.f; es wird mit dem UP-Aufruf nur ein Differenzzeitpunkt geschrieben.

Directory: $(PROGHOME)/public/record/<*system*>/

Record-Status: Record kann vorhanden sein.

Record-Inhalt: Beschreibung eines Differenzzeitpunkts:

① Anzahl Differenzzeitpunkte.
`typ = INTEGER`,
`var = NDZ`.
Diese Zahl wird nur beim Auslesen ermittelt und liefert die Anzahl der in der ausgewählten BDF-Datei des Typs `RI` enthaltenen Differenzzeitpunkte der zu der Datengruppe gehörenden Datensätze.

② Jahr.
`typ = INTEGER`,
`var = DZJAHR(MAXNDZ)`, mit `MAXNDZ` $\geq$ `NDZ`.

③ Monat.
`typ = INTEGER`,
`var = DZMONAT(MAXNDZ)`, mit `MAXNDZ` $\geq$ `NDZ`.

④ Tag.
`typ = INTEGER`,
`var = DZTAG(MAXNDZ)`, mit `MAXNDZ` $\geq$ `NDZ`.

⑤ Stunde.
`typ = INTEGER`,
`var = DZSTUNDE(MAXNDZ)`, mit `MAXNDZ` $\geq$ `NDZ`.

⑥ Minute.
`typ = INTEGER`,
`var = DZMINUTE(MAXNDZ)`, mit `MAXNDZ` $\geq$ `NDZ`.

⑦ Sekunde.
`typ = INTEGER`,
`var = DZSEKUNDE(MAXNDZ)`, mit `MAXNDZ` $\geq$ `NDZ`.

⑧ Nanosekunde.
`typ = INTEGER`,
`var = DZNANOSEKUNDE(MAXNDZ)`, mit `MAXNDZ` $\geq$ `NDZ`.

4.2.13 Info-Recordtyp 012

Lesen und Schreiben der Art der physik. Größen, die in einer BDF-Datei des Typs `DD` als Daten-Records abgespeichert sind. Mit diesen in der BDF-Datei des Typs `RI` abgelegten Informationen ist es möglich, die Art der abgespeicherten Daten näher zu beschreiben.

Auslesen: FORTRAN-Unterprogramm `get012.f`.

Speichern: FORTRAN-Unterprogramm `rec012.f`. Die Angaben zur Art der physikalischen Größen werden aus den schon bekannten Größen `IAKTPG` und `LISTEPG` (siehe Abschnitt 4.2.5) mit Hilfe der in dem Includemodul `phycod_f90` vorhandenen Informationen (siehe Abschnitt 3.2) erzeugt.

Directory: `$(PROGHOME)/public/record/`<*system*>`/`

Record-Status: Record muß vorhanden sein.

Record-Inhalt: Art der abgespeicherten physikalischen Größen:

① Art physik. Größe.
`typ = INTEGER`,
`var = IPTYPE(MAXPG)`, mit `MAXPG` $\geq$ `IAKTPG`.
Mit dieser Zahl wird ein *Pointer* auf eine Position in dem Feld `CPTYP` in der Include-Datei `direkt.h` erzeugt.

1 = `Skalar`.

2 = `Vektor`.

3 = `Tensor`.

`IPTYP` wird häufig das lokale Feld `DATENART(MAXPG)` zur Seite gestellt, welches die vorgenannten Klartextbezeichnungen enthält (siehe hierzu die Unterprogramme `dadrir.f` und `getart.f` aus dem Public-Verzeichnis `record`).

4.2.14 Info-Recordtyp 013

Lesen und Schreiben der Dimension der physik. Größen für die in einer BDF-Datei des Typs `DD` Daten-Records abgespeichert sind. Mit diesen in der BDF-Datei des Typs `RI` abgelegten Informationen ist es möglich, die Art der abgespeicherten Daten näher zu beschreiben.

Auslesen: FORTRAN-Unterprogramm `get013.f`.

Speichern: FORTRAN-Unterprogramm `rec013.f`.

Directory: `$(PROGHOME)/public/record/`<*system*>`/`

Record-Status: Record muß vorhanden sein.

Record-Inhalt: Dimension der abgespeicherten physikalischen Größen:

① Dimension physik. Größe.
`typ = INTEGER`,
`var = LISTEDIM(MAXPG)`, mit `MAXPG` $\geq$ `IAKTPG`.
Hiermit wird die Dimension, d. h. die Anzahl der Komponenten vektorieller und tensorieller Daten festgelegt. Skalare Größen müssen die Dimension 1 erhalten.

4.2.15 Info-Recordtyp 014

Lesen und Schreiben des FORTRAN-Datentyps der in einer BDF-Datei des Typs `DD` abgespeicherten Datenrecords.

Auslesen: FORTRAN-Unterprogramm `get014.f`.

Speichern: FORTRAN-Unterprogramm `rec014.f`.

Directory: `$(PROGHOME)/public/record/`<*system*>`/`

Record-Status: Record muß vorhanden sein.

Record-Inhalt: Beschreibt den FORTRAN-Datentyp, welcher bei der Speicherung der Datenrecords verwendet wird:

① FORTRAN-Datentyp der Records.
`typ = CHARACTER`,
`var = DATENTYP(MAXPG)*3`, mit `MAXPG` $\geq$ `IAKTPG`.

IN = `INTEGER`

RE = `REAL`

DP = `DOUBLE PRECISION`

CX = `COMPLEX`

RI2 = `INTEGER*2` (Normierung für REAL-Daten)

RR2 = `INTEGER*2` (Normierung und Datenersetzung für REAL-Daten)

Daten des Typs `REAL` können auf drei verschiedene Arten abgelegt werden. In der Form `RE` sind sie als gewöhnliche REAL-Zahlen in der BDF-Datei des Typs `DD` enthalten.
In der Form `RI2` werden REAL-Zahlen als normierte, diskretisierte Werte in dem Format `INTEGER*2` gespeichert. Hierbei wird der Wertebereich zwischen Datenmaximum und -minimum gleichmässig in 65534 Intervalle unterteilt und jede REAL-Zahl dem nächstgelegenen diskreten Wert zugeordnet. Hierbei tritt ein Diskretisierungsfehler auf.
Die Form `RR2` entspricht weitestgehend der Form `RI2`, jedoch werden alle Zahlenwerte oberhalb bzw. unterhalb eines wählbaren Grenzwerts vor der Diskretisierung durch geeignete Dummywerte ersetzt.
Bei den Formen `RI2` und `RR2` wird gegenüber `RE` nur der halbe Plattenplatz zur Datenspeicherung benötigt. Weitere Angaben zu Grenzwerten und Diskretisierungsfehler folgen in dem Abschnitt 4.2.18.

4.2.16 Info-Recordtyp 015

Lesen und Schreiben der Zeitabhängigkeit der physik. Größe der in einer BDF-Datei des Typs `DD` abgespeicherten Datenrecords.

Auslesen: FORTRAN-Unterprogramm `get015.f`.

Speichern: FORTRAN-Unterprogramm `rec015.f`. Die Angaben zur Zeitabhängigkeit der physikalischen Größen werden aus den schon bekannten Größen `IAKTPG` und `LISTEPG` (siehe Abschnitt 4.2.5) mit Hilfe der in dem Includemodul `phycod_f90` vorhandenen Informationen (siehe Abschnitt 3.2) erzeugt.

Directory: `$(PROGHOME)/public/record/`<*system*>`/`

Record-Status: Record muß vorhanden sein.

Record-Inhalt: Beschreibt die Zeitabhängigkeit der physikalischen Größen:

① Zeitabhängigkeit physik. Größe.
`typ = INTEGER`,
`var = ZTA(MAXPG)`, mit `MAXPG` $\geq$ `IAKTPG`.

0 = nicht »zeitartig« (z. B. Anzahl).

1 = synoptisch (ein Zeitpunkt oder Ereignis).

2 = synthetisch (z. B. Mittelwert fuer Zeitraum).

4.2.17 Info-Recordtyp 016

Lesen und Schreiben von Name der physik. Größe und Einheit der physik. Größe der in einer BDF-Datei des Typs `DD` abgespeicherten Datenrecords.

Auslesen: FORTRAN-Unterprogramm `get016.f`.

Speichern: FORTRAN-Unterprogramm `rec016.f`. Die Angaben zu Name und Einheit der physikalischen Größen werden aus den schon bekannten Größen `IAKTPG` und `LISTEPG` (siehe Abschnitt 4.2.5) mit Hilfe der in dem Includemodul `phycod_f90` vorhandenen Informationen (siehe Abschnitt 3.2) erzeugt.

Directory: `$(PROGHOME)/public/record/`<*system*>`/`

Record-Status: Record muß vorhanden sein.

Record-Inhalt: Beschreibt Name und Einheit der physikalischen Größen:

① Name physik. Größe.
`typ = CHARACTER`,
`var = CPN(MAXPG)*(40)`, mit `MAXPG` $\geq$ `IAKTPG`.

② Einheit physik. Größe.
`typ = CHARACTER`,
`var = CPU(MAXPG)*(10)`, mit `MAXPG` $\geq$ `IAKTPG`.

4.2.18 Info-Recordtyp 017

Lesen und Schreiben von Informationen über Normierung und Datenersetzung der in einer BDF-Datei des Typs `DD` abgespeicherten Datenrecords, falls diese in den Formen `RI2` oder `RR2` abgespeichert sind (siehe Abschnitt 4.2.15).

Auslesen: FORTRAN-Unterprogramm `get017.f`. Unterprogramm ist noch ohne Funktion.

Speichern: FORTRAN-Unterprogramm `rec017.f`. Unterprogramm ist noch ohne Funktion.

Directory: `$(PROGHOME)/public/record/`<*system*>`/`

Record-Status: Record kann vorhanden sein.

Record-Inhalt: Minimum `RLOW` und Maximum `RHIGH` der Daten, Intervall `RDIF`, Diskretisierungsfehler `RERRMAX`, unterer und oberer Ersetzungswert (`XRMIN` und `XRMAX`) für Normierung und Datenersetzung.

Alle Informationen über Datenmaximum, Datenminimum, Datenintervall und Diskretisierungsfehler sowie zu oberer Ersetzungswert und unterer Ersetzungswert werden direkt mit dem einzelnen Daten-Record in die BDF-Datei des Typs `DD` geschrieben.

4.2.19 Info-Recordtyp 018

Lesen und Schreiben der Anzahl der Datenvariationen physik. Größen der in einer BDF-Datei des Typs `DD` abgelegten physikalischen Größen. Unter >Datenvariation< ist zu verstehen, daß eine physikalische Größe unter gleichem Namen in einer Datei mehrfach auftreten kann. So könnte beispielsweise der Schwebstoffgehalt für mehrere Kornfraktionen gleichzeitig in *einer* Datei vorhanden sein.

Auslesen: FORTRAN-Unterprogramm `get018.f`.

Speichern: FORTRAN-Unterprogramm `rec018.f`.

Directory: `$(PROGHOME)/public/record/`<*system*>`/`

Record-Status: Record muß vorhanden sein.

Record-Inhalt: Beschreibt die Variationen der physikalischen Größen:

① Anzahl Datenvariationen physik. Größe.
`typ = INTEGER`,
`var = LISTEVAR(MAXPG)`, mit `MAXPG` $\geq$ `IAKTPG`.
In der Regel liegen die physikalischen Größen nur in einer Variation vor, so daß `LISTEVAR=1` ist.

4.2.20 Info-Recordtyp 019

Lesen und Schreiben von Zusatzinformationen zu den in einer BDF-Datei des Typs `DD` abgelegten physikalischen Größen. Hierunter fallen all jene Informationen, die nicht in das Schema der restlichen Info-Records passen. Beispielsweise kann die Information über die minimale Wasserbedeckung an den Berechnungspunkten eines Gitters, die dazu benötigt wird, um Trockenfallen und Überfluten zu unterscheiden, als zusätzliche Information in der BDF-Datei des Typs `RI` abgelegt sein.

Auslesen: FORTRAN-Unterprogramm `get019.f`. Es werden mit dem UP-Aufruf alle Zusatzinformationen ausgelesen.

Speichern: FORTRAN-Unterprogramm `rec019.f`. Es wird mit dem UP-Aufruf nur eine Zusatzinformation geschrieben.

Directory: `$(PROGHOME)/public/record/`<*system*>`/`

Record-Status: Record kann vorhanden sein.

Record-Inhalt: Beschreibt verschiedene zusätzliche Informationen:

① Anzahl Zusatzinformationen.
`typ = INTEGER`,
`var = NCX`.
Diese Zahl wird nur beim Auslesen ermittelt und liefert die Anzahl der in der ausgewählten BDF-Datei des Typs `RI` enthaltenen Zusatzinformationen.

② Code zugeordnete physik. Größe.
`typ = INTEGER`,
`var = LXPHYS(MAXNCX)`, mit `MAXNCX` $\geq$ `NCX`.
Jeder Zusatzinformation kann die Code-Nummer einer physikalischen Größe zugeordnet sein, für welche diese Information eine Bedeutung hat oder haben kann.

③ zugeordnete Datenvariation.
`typ = INTEGER`,
`var = LXVARI(MAXNCX)`, mit `MAXNCX` $\geq$ `NCX`.
Jeder Zusatzinformation kann die Nummer der Variation einer physikalischen Größe zugeordnet sein. Diese Zahl muß kleiner oder gleich der Anzahl der für `LXPHYS` maximal zulässigen Datenvariationen physik. Größen sein (siehe Abschnitt 4.2.19).

④ Klartext Zusatzinformation.
`typ = CHARACTER`,
`var = CX(MAXNCX)*40`, mit `MAXNCX` $\geq$ `NCX`.
Hier ist im Klartext die Zusatzinformation enthalten. Als Beispiel soll an dieser Stelle die Codierung der Grenzwassertiefe `HLIMIT` in dem Unterprogramm `x_mkinfo.f` (Public-Verzeichnis `lq2pro`) angegeben werden:
`CX(...) = 'RE#F8.4#HLIMIT=000.0500#m'`
Von links nach rechts erkennt man vier durch # getrennte Teilfelder, welche den Typ der Daten (hier `RE`), die Formatanweisung (hier `F8.4`), die Variable `HLIMIT` mit dem ihr zugewiesenen Wert und die Einheit (hier `m`) enthalten. Mit Hilfe des Unterprogrammes `xzihlimit.f` (Public-Verzeichnis `record`) kann diese Information wieder »decodiert« werden.

Bislang werden folgende Zusatzinformationen (optional) in Dateien des Typs `RI` abgelegt:

- Grenzwassertiefe: `HLIMIT` bezeichnet diejenige Wasserbedeckung an einem Berechnungspunkt, bei deren Unterschreitung dieser als Trockengefallen angesehen werden kann;
- morphodyn. Topographie: mit der Variablen `IMORPHO` wir beschrieben, ob zu den in der Datei des Typs `DD` abgelegten Größen eine zeitvariable morphodynamische Topographie gehört (`imorpho=1`) oder nicht (`imorpho=0`);
- integrale Größen: mit der Variablen `INTMIT` wird angegeben, ob es sich bei den in der Datei des Typs `DD` abgelegten Größen um (querschnittsbezogene) integrale Größen handelt (`intmit=1`) oder nicht (`intmit=0`);
- Intervallgrenzen: die Werte `LOW` und `HIGH` beschreiben den Anfangs- und den Endwert eines Intervalls (z. B. im Zusammenhang mit einer Häufigkeitsverteilung);
- Sektorgrenzen: die Werte `LOW` und `HIGH` beschreiben hier den Anfangs- und Endwinkel eines Sektors (Gradangaben im Uhrzeigersinn);
- konstante Dichte: mit `DENSITY` wird eine im gesamten Wasserkörper konstante Dichte angegeben;
- Schwerebeschleunigung: durch `GRAVITY` wird eine für den gesamten Wasserkörper konstante Schwerebeschleunigung festgelegt;

- konstanter Luftdruck: durch `P0` wird der an der Wasseroberfläche herrschende konstante Luftdruck angegeben;
- konstanter Salzgehalt: durch `SALT` wird ein im gesamten Wasserkörper konstanter Salzgehalt bezeichnet;
- konstante Temperatur: durch `TEMP` wird eine im gesamten Wasserkörper konstante Temperatur beschrieben.

4.2.21 Info-Recordtyp 020

Lesen und Schreiben von Informationen über die Vertikalstruktur einer zumeist zwei- oder dreidimensionalen Modellgeometrie.

Auslesen: FORTRAN-Unterprogramm `get020.f`.

Speichern: FORTRAN-Unterprogramm `rec020.f`.

Directory: `$(PROGHOME)/public/record/`<*system*>`/`

Record-Status: Record sollte vorhanden sein.

Record-Inhalt: Beschreibt die Vertikalstruktur der Modellgeometrie:

① Anzahl Schichtgrenzen.
`typ = INTEGER`,
`var = KMX`.
Diese Zahl beschreibt die Anzahl der Schichtgrenzen, welche die Modellgeometrie in der Vertikalen strukturiert.

② Typ Vertikalstruktur.
`typ = INTEGER`,
`var = TKMX`.

0 = es liegen keine Angaben zur Vertikalstruktur vor.

1 = die Vertikalstruktur ist durch eine Folge von horizontal jeweils in konstanter Tiefe liegende Schichten (z-Schichten) definiert.

2 = die Vertikalstruktur ist durch eine Folge von bodenfolgenden Schichten (Sigma-Schichten) definiert; die Sigma-Tiefe einer Schicht ist dabei jeweils konstant.

③ Schichttiefen.
`typ = REAL`,
`var = H(0:MAXKMX)`, `MAXKMX` $\geq$ `KMX`.
`H(0)` muß die größte in dem Modellgebiet vorkommende Wassertiefe enthalten. Danach folgen `H(1)` ... `H(KMX)`, streng aufsteigend sortiert, von unten nach oben. `H(KMX)` sollte dabei den größtmöglichen Höhenwert enthalten – dieser beträgt üblicherweise –10000 m (Modelltiefen sind nach unten positiv).
Analog kann bei Sigma-Tiefen verfahren werden; allerdings stehen bislang keine Modelle mit Sigma-Tiefen zur Verfügung.

An dieser Stelle muß darauf verwiesen werden, daß Berechnungsergebnisse einer dreidimensionalen Simulationsrechnung typischerweise zwei- *und* dreidimensionalen Charakter haben. So handelt es sich bei der Wasserspiegelauslenkung um eine zweidimensionale Größe ohne Tiefenstruktur, wohingegen die Strömungsgeschwindigkeit eine Vertikalstruktur besitzt. In einer solchen Situation müssen die zwei- und dreidimensionalen Größen unbedingt getrennt in verschiedenen Dateien gespeichert werden. Für den Wasserstand ist dann `KMX=1` zu vereinbaren, während für die Strömungsgeschwindigkeit die tatsächliche Anzahl der Schichten `KMX` verwendet werden muß.

5 Anwenden der FORTRAN77-Schnittstellen

In diesem Kapitel wird beschrieben, wie die »klassischen« FORTRAN77-Schnittstellen in den verschiedenen Phasen der Arbeit mit BDF-Dateien benutzt werden können. Mittlerweile wurden jedoch darüber hinaus mehrere Fortran90-Module erstellt, die ein Arbeiten mit BDF-Daten im Vergleich zu den bislang vorhandenen Schnittstellen weiter vereinfachen; diese werden in Kapitel 6 auf Seite 35 sowie in dem Kapitel 7 auf Seite 54 eingehend beschrieben. Für Neuentwicklungen sollten – soweit wie möglich – diese einfacher zu nutzenden und sichereren Schnittstellen benutzt werden. Zum besseren Verstehen existierender FORTRAN77-Software ist jedoch die Kenntnis der nachfolgend beschriebenen Zusammenhänge unverzichtbar.

5.1 Initialisierungsphase

Die Initialisierungsphase muß in einem Anwendungsprogramm, welches BDF nutzen möchte, allen anderen Aktionen vorausgehen und unmittelbar nach dem Start des Programmes erfolgen. Diese Phase dient im wesentlichen dazu, die in dem Includemodul `phycod_f90` abgelegten sowie die in der Include-Datei `direkt.h` in verschiedenen COMMON-Blöcken abgespeicherten Variablen mit geeigneten Werten vorzubelegen (siehe hierzu nochmals die Abschnitte 3.1 und 4.1). Normalerweise sind die folgenden Initialisierungsschritte zu durchlaufen:

`cerrinit.f`: Dieses Unterprogramm (Public-Verzeichnis `etc`) initialisiert in erster Linie das Feld `CERR` (für Fehlertexte) und belegt `ISPRACHE` mit 1 (Name und Einheit der physikalischen Größen erscheinen dann standardmäßig in deutscher Sprache).

`chocom.f`: Mit diesem Unterprogramm (Public-Verzeichnis `etc`) wird der Computer-Typ dynamisch festgelegt, auf dem das Anwendungsprogramm aktuell ablaufen soll. Diese Auswahl ist deshalb erforderlich, da das Öffnen der BDF-Dateien des Typs `RI` und `DD` – bei beiden handelt es sich um Direktzugriffsdateien – vom Rechnersystem abhängig ist, weil die Recordlänge oftmals unterschiedlich ermittelt wird oder die Wortlängen verschieden sind. Das Unterprogramm speichert daher die Variable `IBYPROWO` (Anzahl der Bytes je Wort) in dem COMMON-Block DDZZ ab. Das Unterprogramm `uniopfi.f` (Public-Verzeichnis `io`) greift dann beim Öffnen von Direktzugriffsdateien auf den gültigen Wert zu. Werden neue Rechnersysteme eingeführt, so müssen diese in `chocom.f` berücksichtigt werden.

`chospr.f`: Die Benutzung dieses Unterprogramms (Public-Verzeichnis `etc`) ist optional. Mit ihm kann der Wert von `ISPRACHE` abgeändert und somit eine andere Sprache

gewählt werden, falls z. B. die englischen Bezeichnungen für Name und Einheit der physikalischen Größen an Stelle der deutschen verwendet werden sollen.

`dirini.f`: Dieses Unterprogramm (Public-Verzeichnis `record`) nimmt eine Initialisierung der Include-Datei direkt.h vor. Variablen und Felder werden mit geeigneten Werten vorbelegt.

`phydef.f`: Das Unterprogramm (Public-Verzeichnis `etc`) steuert die Initialisierung der in Includemodul `phycod_f90` abgelegten, mit `save`-Attribut versehenen Variablen und Felder. Hierbei werden die in dem Abschnitt 3.2 aufgeführten Konfigurationsdateien gelesen, wodurch die Informationen aus diesen in die Variablen und Felder des Includemoduls `phycod_f90` übertragen werden.

`dirdef.f`: Dieses Unterprogramm ruft die Unterprogramme `dirdef1.f`, `dirdef2.f`, `dirdef3.f`, `dirdef4.f` und `dirdef5.f` auf (alle im Public-Verzeichnis `record`). Hierbei werden u. a. die historisch festgelegten Dateitypen definiert und in das in der Include-Datei `direkt.h` abgelegte Feld `CIDCED` eingetragen. Ferner werden feststehende Bezeichnungen, wie z. B. die erlaubten Dateiformen oder Dateizugriffsarten, in den Feldern `CIDFOR`, `CIDACC`, `CIDBEZ`, `CIDPOS` und `CPTYP` festgelegt. Des weiteren wird für jeden zulässigen Typ eines Info-Records die Info-Recordlänge ermittelt. Sollten einmal eines fernen Tages neue Info-Record-Typen eingeführt werden, so ist deren Recordlänge in dem Unterprogramm `dirdef5.f` festzulegen.

Damit sind alle Schritte der Initialisierungsphase genannt. In diesem Zusammenhang sei an dieser Stelle auf den Quellencode des Programms TDKVF verwiesen.

Die vorgenannten Initialisierungen sind nur einmal, unmittelbar nach dem Start eines Anwendungsprogrammes durchzuführen. Wie in dem Abschnitt 4.1.1 dargelegt, können in der Include-Datei `direkt.h` allerdings Informationen über insgesamt `MDFILE` BDF-Dateien gleichzeitig verwaltet werden. Soll daher im weiteren Verlauf eines Anwendungsprogrammes an der Stelle einer schon abgespeicherten Datei die Informationen über eine neue Datei eingespeist werden, weil z. B. die alte Datei nicht mehr benötigt wird, so kann mit Hilfe des Unterprogrammes `dirinip.f` (Public-Verzeichnis `record`) eine gezielte Re-Initialisierung der Variablen und Felder für diese bestimmte Datei-Position vorgenommen werden. Die Verwendung von `dirinip.f` zeigt das Unterprogramm `gen_mkinfo.f` (im Public-Verzeichnis `analyse`).

5.2 Info-Records Lesen und Verarbeiten

In diesem Abschnitt wird das Arbeiten mit Info-Records für Standardanwendungen näher beschrieben, da das Lesen, das standardmäßige Auswerten sowie das Übernehmen der Info-Records in lokale selbstdefinierte Felder und Variablen wesentliche Schritte in (fast) allen Anwendungsprogrammen sind.

5.2.1 Lesen der Info-Records

Das Lesen der Info-Records ist der erste Schritt. Darin werden die in einer BDF-Datei des Typs `RI` abgespeicherten Informationen eingelesen und in die dafür vorgesehenen Felder und Variablen in der Include-Datei `direkt.h` übertragen. Voraussetzung für diesen Schritt

ist die vorherige erfolgreiche Durchführung der in dem Abschnitt 5.1 beschriebenen Initialisierungsphase. Hierfür steht folgendes Unterprogramm bereit:

dadrif.f: Dieses Unterprogramm aus dem Public-Verzeichnis `record` liest zunächst aus der BDF-Datei des Typs `RD` die für das Öffnen der Datei des Typs `RI` benötigte Recordlänge ein. Anschließend werden alle in der Datei des Typs `RI` vorhandenen Info-Records gelesen und in die Variablen und Felder der Include-Datei `direkt.h` geschrieben.

Nach Beendigung dieses Schritts liegen alle für das spätere Öffnen und Lesen der in der zugehörigen BDF-Datei des Typs `DD` abgespeicherten Datenrecords benötigten Informationen ordnungsgemäß in `direkt.h` vor. Die Verwendung von `dadrif.f` zeigt beispielhaft das zu dem Analyseprogramm TDKVF gehörende Unterprogramm `startup.f`.

5.2.2 Auswerten der Info-Records

Nach dem korrekten Speichern von Info-Records in den Variablen und Feldern der Include-Datei `direkt.h` (siehe vorangehenden Abschnitt 5.2.1) kann mit verschiedenen Unterprogrammschnittstellen ein erstes Auswerten der Info-Records vorgenommen werden, um darin gezielt nach bestimmten Informationen zu suchen.

get???.f: Die in dem Abschnitt 4.2 vorgestellten Low-Level Unterprogramme `get000.f` ... `get020.f` (alle im Public-Verzeichnis `record`) ermöglichen ein gezieltes Lesen jedes gültigen Info-Recordtyps.

getsys.f: Dieses Unterprogramm (Public-Verzeichnis `record`) erlaubt ein gezieltes Suchen in den Attributen einer INP-Datei nach dem Vorkommen der Beschreibung einer Systemdatei. Als Ergebnis erhält man im wesentlichen Name und Typ der Systemdatei. Das Unterprogramm `startup.f` von TDKVF zeigt die Anwendung von `getsys.f`.

getzpa.f: Werden Informationen über ein abgespeichertes charakteristisches Datum benötigt (charakteristischer Zeitpunkt), so liefert das Unterprogramm `getzpa.f` (Public-Verzeichnis `record`) für jeden zulässigen Datumstyp (Bezugszeitpunkt, Anfangszeitpunkt und Endzeitpunkt) den gewünschten Wert als Character-String in verschiedenen Formaten.

getozs.f: Sind in den Info-Records Angaben zu Ausgabezeitpunkten enthalten, so liefert dieses Unterprogramm (Public-Verzeichnis `record`) in Analogie zu `getzpa.f` den gewünschten Wert als Character-String in verschiedenen Formaten für jeden beliebigen abgespeicherten Zeitpunkt. Beispielsweise benutzt `chk_inp_d.f` (Public-Verzeichnis `analyse`) dieses Unterprogramm.

dadspq.f: Dieses Unterprogramm aus dem Public-Verzeichnis `record` durchsucht die Info-Records nach dem Vorkommen der Code-Nummer einer physikalischen Größe. Damit kann festgestellt werden, ob eine bestimmte physikalische Größe in einer Datei vorhanden ist oder nicht (zur Definition physikalischer Größen siehe Abschnitt 3.2). Das Unterprogramm `startup.f` von TDKVF zeigt die Anwendung von `dadspq.f`.

xzihlimit.f: Dieses Unterprogramm aus dem Public-Verzeichnis `record` prüft, ob in den Info-Records Zusatzinformationen hinsichtlich der Grenzwassertiefe `HLIMIT` abgelegt sind oder nicht. Das Unterprogramm `startup.f` von TDKVF demonstriert die Verwendung von `xzihlimit.f`.

xzimorpho.f: Dieses Unterprogramm aus dem Public-Verzeichnis `record` prüft, ob in den Info-Records Zusatzinformationen hinsichtlich einer morphodyn. Topographie abgelegt sind oder nicht. Das Unterprogramm `morphocheck.f` in Public-Verzeichnis `record` zeigt das Verwenden von `xzimorpho.f`.

xziintmit.f: Dieses Unterprogramm aus dem Public-Verzeichnis `record` prüft, ob in den Info-Records Angaben enthalten sind, daß es sich bei den dazugehörigen Daten um integrale Größen handelt oder nicht. Unterprogramm `startup.f` von TDKVF demonstriert die Verwendung von `xziintmit.f`.

Diese Aufzählung ist unvollständig. Sie enthält allerdings einige der besonders häufig benutzten Funktionen.

5.2.3 Übernahme der Info-Records

Es ist in vielen Anwendungsprogrammen zweckmäßig oder notwendig, die in der Include-Datei `direkt.h` über den Inhalt einer BDF-Datei des Typs `DD` vorliegenden Informationen in selbstdefinierte Variablen und Felder zu übernehmen. Diese Übertragung kann, wie schon zuvor erwähnt wurde, für jeden Info-Recordtyp einzeln mit Hilfe der Unterprogramme `get000.f`...`get020.f` erfolgen. Dieser Weg ist aber i. d. R. mühsam und fehleranfällig. Daher existiert für die Übernahme der Info-Records eine High-Level Schnittstelle, die eine Übernahme mit Hilfe eines einzigen Unterprogrammaufrufs ermöglicht – allerdings auf Kosten einer langen Parameterliste. Voraussetzung für die Anwendung dieser High-Level Schnittstelle ist, daß zuvor sowohl die Initialisierungsphase (siehe Abschnitt 5.1) wie auch das Lesen der Info-Records (siehe Abschnitt 5.2.1) fehlerfrei durchgeführt wurden.

dadrir.f: Dieses Unterprogramm aus dem Public-Verzeichnis `record` überträgt alle gewünschten Felder und Variablen aus `direkt.h` in selbstdefinierte lokale Größen. Die Verwendung von `dadrir.f` ist in dem Programm TDKVF zu sehen. Intern greift das Unterprogramm `dadrir.f` auf die Low-Level Unterprogramme `get000.f` bis `get020.f` zurück.

5.3 Speicherung der Daten

Wie in Kapitel 2 schon früher erwähnt wurde, können in BDF-Dateien des Typs `DD` ganzzahlige, reellwertige, komplexe oder doppeltgenaue Zahlen für skalare, vektorielle und tensorielle Größen abgelegt werden. Die Daten werden dabei komponentenweise als einzelne Records geschrieben oder gelesen.

5.3.1 Dimensionierung der Felder

In dem jeweiligen Anwendungsprogramm müssen zur Aufnahme der physikalischen Größen die entsprechenden Speicherfelder in geeigneter Größe definiert werden. Hierbei ist

zusätzlich zu beachten, daß die Dimensionierung der FORTRAN-Felder von der Datensatzart abhängig ist, die in der BDF-Datei des Typs DD abgespeichert sind. Derzeit sind folgende Formen bekannt:

ZEITREIHE: Es sind Daten vom Typ >äquidistante Zeitreihe< im Speicherformat >VECTOR< abgespeichert.

SYNOPTISCH: Der Typ der abgespeicherten Daten ist ein >synoptischer Datensatz<. Von diesem können beliebig viele Ausgabezeitpunkte enthalten sein. Die Daten werden im Speicherformat >TICAD< abgelegt.

EINZELANALYSE: Die abgespeicherten Daten sind vom Typ >Ergebnisse Einzelanalyse<. Die Daten liegen in dem Speicherformat >TICAD< vor.

DIFFERENZANALYSE: Die abgespeicherten Daten sind vom Typ >Ergebnisse Differenzanalyse<. Die Daten liegen in dem Speicherformat >TICAD< vor.

Die Art der abgespeicherten Datensätze kann mit Hilfe des High-Level Unterprogramms `gen_dad_typ.f` (Public-Verzeichnis `record`) abgefragt werden. Es folgen Angaben zur Dimensionierung der Felder. Die zur Aufnahme der Berechnungsergebnisse benötigten Felder sind wie folgt zu deklarieren:

```
C     Integer-Groessen
      INTEGER
     +     ILOCAL(MAXIN), IGLOBAL(MAXIN1,MAXIN2,MAXIN3)
      INTEGER*2
     +     I2FELD(MAXI2)
C     Reelle Groessen
      REAL
     +     RLOCAL(MAXRE), RGLOBAL(MAXRE1,MAXRE2,MAXRE3)
C     Doppeltgenaue Groessen
      DOUBLE PRECISION
     +     DLOCAL(MAXDP), DGLOBAL(MAXDP1,MAXDP2,MAXDP3)
C     komplexe Groessen
      COMPLEX
     +     CLOCAL(MAXCX), CGLOBAL(MAXCX1,MAXCX2,MAXCX3)
```

Die Felder mit den Namen `ILOCAL`, `RLOCAL`, `DLOCAL` und `CLOCAL` (`I2FELD` für den Fall, daß eine Normierung der Daten gegeben ist) dienen beim Lesen oder Schreiben zum Speichern der Daten (bei vektoriellen oder tensoriellen Größen wird immer nur eine Komponente je Datensatz beim Schreiben / Lesen gespeichert). Demgegenüber nehmen die Felder namens `IGLOBAL`, `RGLOBAL`, `DGLOBAL` und `CGLOBAL` den gesamten Ergebniszustand auf, also ggf. alle Komponenten und ggf. auch alle Datenvariationen. Üblicherweise wird die Dimension der Felder durch folgende Größen maßgeblich bestimmt:

`MAXDIM`: Maximale Dimension der Ergebnisse. Liegen auch vektorielle Daten vor, so ist diese Größe um 1 größer als die Anzahl der Komponenten, da für vektorielle Größen als zusätzliche Komponente der Betrag oder die Betragsdifferenz (Differenz zwischen zwei Analyseergebnissen) als letzter Wert in der entsprechenden Position eingespeichert wird.

MAXKNO: Maximale Anzahl der Knoten. Dies entspricht bei Ergebnissen zweidimensionaler Modellrechnungen in der Regel der Anzahl der Gitterpunkte in der Systemdatei. Bei dreidimensionalen Modellrechnungen ist diese Zahl stark von der Vertikalstruktur abhängig (Anzahl der Vertikalschichten).

MAXVAR: Maximale Anzahl der Datenvariationen physik. Größen (siehe hierzu Abschnitt 4.2.19).

MAXLZR: Maximale Länge der äquidistanten Zeitreihen (siehe hierzu Abschnitt 4.2.7).

Falls die Daten in dem Speicherformat >TICAD< abgelegt sind, so sind die Feldgrenzen wie folgt festzulegen:

```
MAX??  = MAXKNO
MAX??1 = MAXDIM
MAX??2 = MAXKNO
MAX??3 = MAXVAR
```

Damit kann jeweils ein Ergebniszustand für das Gesamtgebiet abgespeichert werden. Liegen die Daten hingegen in dem Speicherformat >VECTOR< vor, so sind die Feldgrenzen wie folgt festzulegen:

```
MAX??  = MAXLZR
MAX??1 = MAXLZR
MAX??2 = MAXDIM
MAX??3 = MAXVAR
```

Damit kann jeweils eine Zeitreihe abgespeichert werden. Abschließender Hinweis: Liegen keine komplexen oder keine doppeltgenauen Ergebnisse vor, so kann Memory dadurch eingespart werden, daß die entsprechenden Felder, z. B. `CGLOBAL`, jeweils mit der Dimension 1 deklariert werden.

5.4 Daten-Records Lesen

Das Lesen des in einer BDF-Datei des Typs `DD` abgelegten Datensatzes zerfällt in die Teilschritte Datei öffnen, Recordnummer berechnen, Daten-Record lesen und Datei schließen. Werden mehrere Daten-Records hintereinander aus derselben Datei gelesen werden, so ist es natürlich zweckmäßig, daß diese nur einmal geöffnet und auch nur einmal geschlossen wird. Für das im Anschluß beschriebene Vorgehen wird vorausgesetzt, daß die Info-Records der dazugehörenden BDF-Datei des Typs `RI` ordnungsgemäß gelesen wurden (siehe auch Abschnitt 5.2.1).

5.4.1 Daten-Datei Öffnen

Für das Öffnen einer Daten-Datei steht folgendes Unterprogramm zur Verfügung:

`dadodf.f`: Dieses Unterprogramm aus dem Public-Verzeichnis `record` öffnet unter Verwendung der in der Include-Datei `direkt.h` abgespeicherten Informationen die BDF-Datei des Typs `DD` (Öffnen einer OUT-Datei). `dadodf.f` wird beispielsweise von dem Programm TDKVF benutzt.

Das Unterprogramm dadodf.f enthält den Parameter STA, welcher die Werte old oder unknown annehmen darf. Dieser Parameter muß beim späteren Schließen der Datei mit dem Unterprogramm dadcdf.f den gleichen Wert aufweisen.

5.4.2 Daten-Datei Schließen

Für das Schließen einer Daten-Datei steht folgendes Unterprogramm zur Verfügung:

dadcdf.f: Dieses Unterprogramm aus dem Public-Verzeichnis record schließt eine zuvor mit dem Unterprogramm dadodf.f geöffnete BDF-Datei des Typs DD. Zur Verwendung von dadcdf.f siehe das Programm TDKVF.

Das Unterprogramm dadcdf.f enthält den Parameter STA. Dieser Parameter muß beim Schließen den beim Öffnen der Datei benutzten Wert aufweisen.

5.4.3 Recordnummer berechnen

Bevor ein Datensatz von einer BDF-Datei des Typs DD gelesen oder in diese geschrieben werden kann, muß die Recordnummer des Datensatzes bekannt sein. Für die Berechnung der Recordnummer werden in Abhängigkeit von den zulässigen Speicherformaten (>TICAD< und >VECTOR<) folgende Informationen benötigt:

① Code-Nummer physik. Größe
typ = INTEGER,
var = IPHYS.
Code-Nummer der physikalischen Größe, für welche Daten gelesen werden sollen. Für diese Größe muß unabhängig von der Datensatzart immer ein sinnvoller Wert angegeben werden.

② Knoten-Nummer
typ = INTEGER,
var = ACTNODE, 1 ... IAKTKNO.
Nummer des Knotens (Gitterpunktes), für den eine Zeitserie gelesen werden soll. Für diese Größe muß bei folgenden Datensatzarten ein sinnvoller Wert angegeben werden: äquidistante Zeitreihe.

③ Knoten-Nummer
typ = INTEGER,
var = ACTTIMESTEP.
Nummer des Ausgabezeitpunktes, für den ein Datensatz gelesen werden soll. Für diese Größe muß beim Lesen von Ereignissen oder Zeitpunkten immer dann ein sinnvoller Wert angegeben werden, wenn folgende Datensatzarten gelesen werden: synoptischer Datensatz, Ergebnisse Einzelanalyse und Ergebnisse Differenzanalyse.

Liegen die geforderten Informationen vor, so steht für das Berechnen der Recordnummer des gewünschten Datensatzes folgendes Unterprogramm zur Verfügung:

dadcr.f: Dieses Unterprogramm (Public-Verzeichnis record) liefert als Ergebnis die Recordnummer IREC, die für das spätere Lesen der Daten mit dem Unterprogramm dadorr.f benötigt wird.

Das Unterprogramm dadcr.f benutzt seinerseits die Unterprogramme dadcrt.f (für die Datensatzart >äquidistante Zeitreihe<) und dadcrd.f (für alle anderen Datensatzarten) zum Berechnen der Recordnummer.

5.4.4 Record lesen

Unter der Voraussetzung, daß die Berechnung der Recordnummer IREC zuvor korrekt erfolgte (siehe vorangehenden Abschnitt), kann das Lesen eines Daten-Records mit Hilfe der High-Level Unterprogrammschnittstelle dadorr.f erfolgen:

dadorr.f: Dieses Unterprogramm (Public-Verzeichnis record) liest in Abhängigkeit vom FORTRAN-Datentyp in dem die abgespeicherten Daten vorliegen das Ergebnis in eines der zur Speicherung der Ergebnisse bereitstehen Felder. Die Verwendung der Unterprogramme dadcr.f und dadorr.f zeigt das Unterprogramm reflzkwf.f (in dem Public-Verzeichnis analyse).

Daten des Typs IN werden in IGLOBAL zurückgegeben. Demgegenüber dient das Feld RGLOBAL zur Aufnahme von Ergebnissen der Datentypen RE, RI2 und RR2. Komplexe Daten des Typs CX werden in CGLOBAL und doppeltgenaue des Typs DP in DGLOBAL abgespeichert.

5.5 Info-Records Schreiben

In diesem Abschnitt wird zu dem Thema Arbeiten mit Info-Records für Standardanwendungen näher beschrieben, welche Schritte für das Schreiben eigener Info-Records erforderlich sind. In Anwendungsprogrammen, die selbst neue BDF-Dateien erstellen, muß vor dem Schreiben der ersten Daten-Records die zu der BDF-Datei des Typs DD korrespondierende BDF-Datei des Typs RI erstellt worden sein. Grundsätzlich müssen vor dem Erzeugen und Beschreiben der Datei des Typs RI alle über die abzuspeichernden Daten benötigten Informationen in den Variablen und Feldern der Include-Datei direkt.h korrekt abgelegt worden sein.

5.5.1 Übertragen der Info-Records

Das Übertragen der Info-Records in die Variablen und Felder der Include-Datei direkt.h ist der erste Schritt. Prinzipiell kann zu diesem Zweck auf die Low-Level Unterprogramme rec000.f ...rec020.f (siehe hierzu den Abschnitt 4.2) zurückgegriffen werden. Empfohlen wird allerdings, auf die High-Level Schnittstelle dadgir.f zurückzugreifen. Dieses Unterprogramm führt die zu dem in dem Abschnitt 5.2.3 aufgeführten Unterprogramm dadrir.f gegensätzlichen Operationen aus. Voraussetzung für die Anwendung von dadgir.f ist, daß alle benötigten Informationen in lokalen selbstdefinierten Feldern abgelegt sind.

dadgir.f: Dieses Unterprogramm aus dem Public-Verzeichnis record nimmt eine Übertragung der in lokalen selbstdefinierten Feldern abgelegten datenrelevanten Informationen in die Variablen und Felder der Include-Datei direkt.h vor.

Siehe hierzu das Zusammenspiel der Unterprogramme gen_mkinfo.f und tdk_mkinfo.f (beide in dem Public-Verzeichnis analyse).

5.5.2 Schreiben der Info-Records

Wurden alle relevanten Informationen über die abzuspeichernden Daten einmal in die Variablen und Felder der Include-Datei `direkt.h` übertragen, so kann das Schreiben der Info-Records in die BDF-Datei des Typs `RI` ausgeführt werden:

`dadgif.f`: Dieses Unterprogramm aus dem Public-Verzeichnis `record` nimmt zunächst eine Berechnung der Recordlänge der in die BDF-Datei des Typs `RI` zu schreibenden Info-Records vor, und schreibt diese Angaben in eine BDF-Datei des Typs `RD`. Anschließend werden alle zu erzeugenden Info-Records mit Hilfe der Low-Level Unterprogramme `rec00.f` ...`rec020.f` in die Info-Datei des Typs `RI` geschrieben. Öffnen und Schließen aller Dateien wird dabei automatisch durchgeführt.

Als Beispiel siehe das Unterprogramm `tdk_mkinfo.f` (Public-Verzeichnis `analyse`).

5.6 Daten-Records Schreiben

Das Schreiben eines Daten-Records in eine BDF-Datei des Typs `DD` zerfällt in die Teilschritte Datei öffnen, Recordnummer berechnen, Daten-Record schreiben und Datei schließen. Werden mehrere Daten-Records hintereinander in dieselbe Datei geschrieben, so kann das wiederholte Öffnen und Schließen entfallen. Zum Schreiben der Daten-Records ist es erforderlich, daß zuvor die Info-Records in die korrespondierende Datei des Typs `RI` geschrieben werden (siehe Abschnitt 5.5).

Das Öffnen und Schließen einer Daten-Datei, sowie die Berechnung der Recordnummer erfolgen vollständig analog zu den in Abschnitt 5.4 gegebenen Erläuterungen. Auf eine detaillierte Darstellung dieser Aktionen wird daher an dieser Stelle verzichtet.

5.6.1 Record schreiben

Unter der Voraussetzung, daß die Berechnung der Recordnummer `IREC` zuvor korrekt erfolgte (siehe Abschnitt 5.4.3), kann der benötigte Datensatz mit Hilfe der High-Level Unterprogrammschnittstelle `dadowr.f` in die BDF-Datei des Typs `DD` geschrieben werden:

`dadowr.f`: Dieses Unterprogramm (Public-Verzeichnis `record`) schreibt in Abhängigkeit vom FORTRAN-Datentyp in dem die abgespeicherten Daten vorliegen das Ergebnis in die Daten-Datei des Typs `DD`.

Daten des Typs `IN` werden in `IGLOBAL` übergegeben. Demgegenüber dient das Feld `RGLOBAL` zur Aufnahme der Datentypen `RE`, `RI2` und `RR2`. Komplexe Daten des Typs `CX` werden in `CGLOBAL` und doppeltgenaue in `DGLOBAL` abgespeichert. Ein Beispiel für die Anwendung von `dadowr.f` zeigt das in dem Public-Verzeichnis `analyse` vorhandene Unterprogramm `anawar2.f`.

6 Selbstdefinierte Fortran90-Datentypen

Mit den in Abschnitt 4 vorgestellten Info-Records können BDF-Daten vollständig beschrieben werden. Allerdings wird zur Aufnahme der Informationen unter FORTRAN77 eine größere Zahl Variablen und Felder benötigt. Unter Fortran90 können diese zu selbstdefinierten Datentypen zusammenfaßt werden. Auf diesem Wege wird die Deklaration der

Größen wesentlich einfacher und vor allem sicherer. Nachfolgend werden alle existierenden Fortran90-Datentypen zur Aufnahme der Informationen über BDF-Daten wie auch zum Speichern der Daten selbst vorgestellt.

6.1 Datentypen für BDF-Informationen

6.1.1 Datentyp `t_bdfazr`

Alle in einer BDF-Datei des Typs `RI` enthaltenen Informationen zur Beschreibung einer äquidistanten Zeitreihe (Charakterisierung Zeitreihe) werden in dem selbstdefinierten Datentyp `t_bdfazr` zusammengefaßt.

Moduldatei: `mod_bdf_azr.f90`.

Directory: `$(PROGHOME)/public/datatypes/`<*system*>`/`.

Modulname: `bdfazr`.

Datentyp: `t_bdfazr` (Charakterisierung Zeitreihe).

Inhalt: entspricht Info-Record Nr. 006 »Charakterisierung Zeitreihe« (siehe hierzu Unterabschnitt 4.2.7 auf Seite 14).

Komponenten:

① Anfangszeit Sekunden.
`typ = INTEGER`,
`nam = ANFANGSZEIT`.

② Anfangszeit Nanosekunden.
`typ = INTEGER`,
`nam = NANOANFANG`.

③ Zeitschritt Sekunden.
`typ = INTEGER`,
`nam = ZEITSCHRITT`.

④ Zeitschritt Nanosekunden.
`typ = INTEGER`,
`nam = NANOSCHRITT`.

⑤ Anzahl Stützstellen.
`typ = INTEGER`,
`nam = ANZAHL`.

Hinweis: Eine ausführliche Beschreibung der Bedeutung der Komponenten befindet sich in dem Unterabschnitt 4.2.7 auf Seite 14.

Modulunterprogramme:

① `init_bdfazr`.
Initialisieren der Komponenten.

② `check_valid_bdfazr`.
Prüfen der Gültigkeit der Komponenten.

6.1.2 Datentyp t_bdfbp

Alle in einer BDF-Datei des Typs RI enthaltenen Informationen zur Beschreibung besonderer Positionen (Positionsbezeichnungen: Bezugsposition und Referenzposition) werden in dem selbstdefinierten Datentyp t_bdfbp zusammengefaßt.

Moduldatei: mod_bdf_bp.f90.

Directory: $(PROGHOME)/public/datatypes/<*system*>/.

Modulname: bdfbp.

Datentyp: t_bdfbp (Positionsbezeichnungen).

Inhalt: enthält alle Info-Records Nr. 008 »Positionsbezeichnung« abgelegten Informationen (siehe Unterabschnitt 4.2.9 auf Seite 16).

Komponenten:

① Anzahl Positionsbezeichnungen.
typ = INTEGER,
nam = ANZAHL (entspricht NBP).

② Code Positionsbezeichnung.
typ = INTEGER, POINTER, DIMENSION(:),
nam = TYP (entspricht BPTYP),
dynamisches Feld.

③ Knotennummer Position.
typ = INTEGER, POINTER, DIMENSION(:),
nam = NODE (entspricht BPNODE),
dynamisches Feld.

④ x-Koordinate Position (in m).
typ = REAL, POINTER, DIMENSION(:),
nam = X (entspricht XBP),
dynamisches Feld.

⑤ y-Koordinate Position (in m).
typ = REAL, POINTER, DIMENSION(:),
nam = Y (entspricht YBP),
dynamisches Feld.

⑥ z-Koordinate Position (in m).
typ = REAL, POINTER, DIMENSION(:),
nam = Z (entspricht ZBP),
dynamisches Feld.

Hinweis: Eine ausführliche Beschreibung der Bedeutung der Komponenten befindet sich in dem Unterabschnitt 4.2.9 auf Seite 16.

Modulunterprogramme:

① init_bdfbp.
Initialisieren der Komponenten.

② `read_bdfbp_anzahl`.
Liest die Anzahl `ANZAHL` (entspricht `NBP`) der in einer BDF-Datei des Typs `RI` enthaltenen Positionsbeschreibungen ein, um z. B. später die dynamischen Felder geeignet dimensionieren zu können.

③ `alloc_bdfbp`.
Allokieren der dynamischen Feldkomponenten des Datentyps.

④ `dealloc_bdfbp`.
Deallokieren der dynamischen Feldkomponenten des Datentyps.

⑤ `check_valid_bdfbp`.
Prüfen der Gültigkeit der Komponenten.

6.1.3 Datentyp `t_bdfco`

Alle in einer BDF-Datei des Typs `RI` enthaltenen Informationen, die im Zusammenhang mit Kommentaren stehen, werden in dem selbstdefinierten Datentyp `t_bdfco` zusammengefaßt.

Moduldatei: `mod_bdf_co.f90`.

Directory: `$(PROGHOME)/public/datatypes/`<*system*>`/`.

Modulname: `bdfco`.

Datentyp: `t_bdfco` (Kommentare).

Inhalt: enthält alle in Info-Records Nr. 005 »Kommentar« abgelegten Informationen (siehe Unterabschnitt 4.2.6 auf Seite 13).

Komponenten:

① Anzahl Kommentare.
`typ = INTEGER`,
`nam = ANZAHL` (entspricht `NCO`).

② Klartext Kommentarzeile.
`typ = CHARACTER (LEN=80), POINTER, DIMENSION(:)`,
`nam = CO`,
dynamisches Feld.

Hinweis: Eine ausführliche Beschreibung der Bedeutung der Komponenten befindet sich in dem Unterabschnitt 4.2.6 auf Seite 13.

Modulunterprogramme:

① `init_bdfco`.
Initialisieren der Komponenten.

② `read_bdfco_anzahl`.
Liest die Anzahl `ANZAHL` (entspricht `NCO`) der in einer BDF-Datei des Typs `RI` enthaltenen Kommentare ein, um z. B. später die dynamischen Felder geeignet dimensionieren zu können.

③ `alloc_bdfco`.
Allokieren der dynamischen Feldkomponenten des Datentyps.

④ `dealloc_bdfco`.
Deallokieren der dynamischen Feldkomponenten des Datentyps.

⑤ `check_valid_bdfco`.
Prüfen der Gültigkeit der Komponenten.

6.1.4 Datentyp `t_bdfdinf`

Alle in einer BDF-Datei des Typs `RI` enthaltenen Informationen, die im Zusammenhang mit der Beschreibung verschiedener Dateien dort vorhanden sind (INP-Dateibeschreibung und OUT-Dateibeschreibung), werden in dem selbstdefinierten Datentyp `t_bdfdinf` zusammengefaßt. Ein- und Ausgabedateicharakterisierungen müssen in getrennten Variablen des Typs `t_bdfdinf` abgelegt werden.

Moduldatei: `mod_bdf_dinf.f90`.

Directory: `$(PROGHOME)/public/datatypes/`<*system*>`/`.

Modulname: `bdfdinf`.

Datentyp: `t_bdfdinf` (Dateibeschreibungen).

Inhalt: enthält alle in den Info-Records Nr. 001 »INP-Dateibeschreibung« (siehe 4.2.2 auf Seite 10) sowie Nr. 007 »OUT-Dateibeschreibung« (siehe 4.2.8 auf Seite 15) abgespeicherten Informationen.

Komponenten:

① Anzahl INP-Dateien *oder* Anzahl OUT-Dateien.
`typ = INTEGER`,
`nam = ANZAHL` (entspricht `NIF` *oder* `NOF`).

② Code INP-Dateiformat *oder* Code OUT-Dateiformat.
`typ = INTEGER, POINTER, DIMENSION(:)`,
`nam = FORM` (entspricht `IFFORM` *oder* `OFFORM`),
dynamisches Feld.

③ Code INP-Dateizugriff *oder* Code OUT-Dateizugriff.
`typ = INTEGER, POINTER, DIMENSION(:)`,
`nam = ACC` (entspricht `IFACC` *oder* `OFACC`),
dynamisches Feld.

④ Code INP-Dateityp *oder* Code OUT-Dateityp.
`typ = INTEGER, POINTER, DIMENSION(:)`,
`nam = CIDCED` (entspricht `IFCIDCED` *oder* `OFCIDCED`),
dynamisches Feld.

⑤ Recordlänge OUT-Datei.
`typ = INTEGER, POINTER, DIMENSION(:)`,
`nam = RECL` (entspricht `OFRECL`),

dynamisches Feld.
Hinweis: Dieses dynamische Feld wird nur für Ausgabedateien `DINFTYP` = `OF` allokiert.

⑥ INP-Dateiname *oder* OUT-Dateiname.
`typ = CHARACTER (LEN=80), POINTER, DIMENSION(:)`,
`nam = CFILE` (entspricht `CIF` *oder* `COF`),
dynamisches Feld.

⑦ Art der Datei.
`typ = CHARACTER (LEN=2)`,
`nam = DINFTYP`.
Damit wird angegeben, welche Art Dateien charakterisiert werden:

`IF` = Eingabedateien;

`OF` = Ausgabedateien.

Hinweis: Eine ausführliche Beschreibung der Bedeutung der Komponenten befindet sich in den Unterabschnitten 4.2.2 auf Seite 10 sowie 4.2.8 auf Seite 15.

Modulunterprogramme:

① `init_bdfdinf`.
Initialisieren der Komponenten.

② `read_bdfdinf_anzahl`.
Liest die Anzahl `ANZAHL` (entspricht `NIF` *oder* `NOF`) der in einer BDF-Datei des Typs `RI` enthaltenen Beschreibungen für Ein- *oder* Ausgabedateien, in Abhängigkeit vom aktuellen Wert der Komponente `DINFTYP`.

③ `alloc_bdfdinf`.
Allokieren der dynamischen Feldkomponenten des Datentyps.

④ `dealloc_bdfdinf`.
Deallokieren der dynamischen Feldkomponenten des Datentyps.

⑤ `check_valid_bdfdinf`.
Prüfen der Gültigkeit der Komponenten.

6.1.5 Datentyp `t_bdfpg`

Alle in einer BDF-Datei des Typs `RI` enthaltenen Informationen, die im Zusammenhang mit der ausführlichen Beschreibung von in der dazugehörenden BDF-Datei des Typs `DD` abgespeicherten physikalischen Größen stehen (Art der physik. Größen, Datenvariationen physik. Größen, Dimension der physik. Größen, Einheit der physik. Größe, FORTRAN-Datentyp, Liste der physik. Größen, Name der physik. Größe und Zeitabhängigkeit der physik. Größe), werden in dem selbstdefinierten Datentyp `t_bdfpg` zusammengefaßt.

Moduldatei: `mod_bdf_pg.f90`.

Directory: `$(PROGHOME)/public/datatypes/`<*system*>`/`.

Modulname: `bdfpg`.

Datentyp: `t_bdfpg` (physikalische Größen).

Inhalt: enthält alle in den Info-Records Nr. 004 »Liste der physik. Größen« (siehe 4.2.5 auf Seite 13), Nr. 012 »Art der physik. Größen« (siehe 4.2.13 auf Seite 20), Nr. 013 »Dimension der physik. Größen« (siehe 4.2.14 auf Seite 21), Nr. 014 »FORTRAN-Datentyp« (siehe 4.2.15 auf Seite 21), Nr. 015 »Zeitabhängigkeit der physik. Größe« (siehe 4.2.16 auf Seite 22), Nr. 016 »Name der physik. Größe« und »Einheit der physik. Größe« (siehe 4.2.17 auf Seite 23) sowie Nr. 018 »Datenvariationen physik. Größen« (siehe 4.2.19 auf Seite 24) abgelegten Informationen.

Komponenten:

① Anzahl physik. Größen.
`typ = INTEGER`,
`nam = ANZAHL` (entspricht `IAKTPG`).

② Code physik. Größen.
`typ = INTEGER, POINTER, DIMENSION(:)`,
`nam = LISTEPG`,
dynamisches Feld.
Hinweis: Weitere Informationen zu dieser Komponente in Unterabschnitt 4.2.5 auf Seite 13.

③ Art physik. Größe.
`typ = INTEGER, POINTER, DIMENSION(:)`,
`nam = IPTYPE`,
dynamisches Feld.
Hinweis: Weitere Informationen zu dieser Komponente in Unterabschnitt 4.2.13 auf Seite 20.

④ Dimension physik. Größe.
`typ = INTEGER, POINTER, DIMENSION(:)`,
`nam = LISTEDIM`,
dynamisches Feld.
Hinweis: Weitere Informationen zu dieser Komponente in Unterabschnitt 4.2.14 auf Seite 21.

⑤ Zeitabhängigkeit physik. Größe.
`typ = INTEGER, POINTER, DIMENSION(:)`,
`nam = ZTA`,
dynamisches Feld.
Hinweis: Weitere Informationen zu dieser Komponente in Unterabschnitt 4.2.16 auf Seite 22.

⑥ Anzahl Datenvariationen physik. Größe.
`typ = INTEGER, POINTER, DIMENSION(:)`,
`nam = LISTEVAR`,
dynamisches Feld.
Hinweis: Weitere Informationen zu dieser Komponente in Unterabschnitt 4.2.19 auf Seite 24.

⑦ FORTRAN-Datentyp der Records.
`typ = CHARACTER (LEN=3), POINTER, DIMENSION(:) :: DATENTYP`,

nam = DATENTYP,
dynamisches Feld.
Hinweis: Weitere Informationen zu dieser Komponente in Unterabschnitt 4.2.15 auf Seite 21.

⑧ Art physik. Größe.
typ = CHARACTER (LEN=6), POINTER, DIMENSION(:) :: DATENART,
nam = DATENART,
dynamisches Feld.
Hinweis: Weitere Informationen zu dieser Komponente in Unterabschnitt 4.2.13 auf Seite 20.

⑨ Name physik. Größe.
typ = CHARACTER (LEN=40), POINTER, DIMENSION(:),
nam = CPN,
dynamisches Feld.
Hinweis: Weitere Informationen zu dieser Komponente in Unterabschnitt 4.2.17 auf Seite 23.

⑩ Einheit physik. Größe.
typ = CHARACTER (LEN=10), POINTER, DIMENSION(:),
nam = CPU,
dynamisches Feld.
Hinweis: Weitere Informationen zu dieser Komponente in Unterabschnitt 4.2.17 auf Seite 23.

Modulunterprogramme:

① init_bdfpg.
Initialisieren der Komponenten.

② read_bdfpg_anzahl.
Liest die Anzahl ANZAHL (entspricht IAKTPG) der in einer BDF-Datei des Typs RI enthaltenen physikalischen Größen ein, um z. B. später die dynamischen Felder geeignet dimensionieren zu können.

③ alloc_bdfpg.
Allokieren der dynamischen Feldkomponenten des Datentyps.

④ dealloc_bdfpg.
Deallokieren der dynamischen Feldkomponenten des Datentyps.

⑤ check_valid_bdfpg.
Prüfen der Gültigkeit der Komponenten.

6.1.6 Datentyp t_bdfvs

Alle (gegebenenfalls) in einer BDF-Datei des Typs RI enthaltenen Informationen zur Beschreibung der Vertikalstruktur werden in dem selbstdefinierten Datentyp t_bdfvs zusammengefaßt.

Moduldatei: mod_bdf_vs.f90.

Directory: `$(PROGHOME)/public/datatypes/`<*system*>`/`.

Modulname: `bdfvs`.

Datentyp: `t_bdfvs` (Vertikalstruktur).

Inhalt: entspricht Info-Record Nr. 020 »Vertikalstruktur« (siehe Unterabschnitt 4.2.21 auf Seite 26).

Komponenten:

① Anzahl Schichtgrenzen.
`typ = INTEGER`,
`nam = KMX`.

② Typ Vertikalstruktur.
`typ = INTEGER`,
`nam = TKMX`.

③ Schichttiefen.
`typ = REAL, POINTER, DIMENSION(:)`,
`nam = H`,
dynamisches Feld.

Hinweis: Eine ausführliche Beschreibung der Bedeutung der Komponenten befindet sich in dem Unterabschnitt 4.2.21 auf Seite 26.

Modulunterprogramme:

① `init_bdfvs`.
Initialisieren der Komponenten.

② `read_bdfvs_anzahl`.
Ermittelt die Anzahl der Schichten `KMX`, um z. B. später die dynamischen Felder geeignet dimensionieren zu können.

③ `alloc_bdfvs`.
Allokieren der dynamischen Feldkomponenten des Datentyps.

④ `dealloc_bdfvs`.
Deallokieren der dynamischen Feldkomponenten des Datentyps.

⑤ `check_valid_bdfvs`.
Prüfen der Gültigkeit der Komponenten.

6.1.7 Datentyp `t_bdfzinf`

Alle (gegebenenfalls) in einer BDF-Datei des Typs `RI` enthaltenen Zusatzinformationen werden in dem selbstdefinierten Datentyp `t_bdfzinf` zusammengefaßt.

Moduldatei: `mod_bdf_zinf.f90`.

Directory: `$(PROGHOME)/public/datatypes/`<*system*>`/`.

Modulname: `bdfzinf`.

Datentyp: `t_bdfzinf` (Zusatzinformationen).

Inhalt: entspricht Info-Record Nr. 019 »Zusatzinformation« (siehe Unterabschnitt 4.2.20 auf Seite 24).

Komponenten:

① Anzahl Zusatzinformationen.
`typ = INTEGER`,
`nam = ANZAHL` (entspricht `NCX`).

② Code zugeordnete physik. Größe.
`typ = INTEGER, POINTER, DIMENSION(:)`,
`nam = LXPHYS`,
dynamisches Feld.

③ zugeordnete Datenvariation.
`typ = INTEGER, POINTER, DIMENSION(:)`,
`nam = LXVARI`,
dynamisches Feld.

④ Klartext Zusatzinformation.
`typ = CHARACTER (LEN=80), POINTER, DIMENSION(:)`,
`nam = CX`,
dynamisches Feld.

Hinweis: Eine ausführliche Beschreibung der Bedeutung der Komponenten befindet sich in dem Unterabschnitt 4.2.20 auf Seite 24.

Modulunterprogramme:

① `init_bdfzinf`.
Initialisieren der Komponenten.

② `read_bdfzinf_anzahl`.
Ermittelt die Anzahl der zusätzlichen Informationen `ANZAHL` (entspricht `NCX`), um z. B. später die dynamischen Felder geeignet dimensionieren zu können.

③ `alloc_bdfzinf`.
Allokieren der dynamischen Feldkomponenten des Datentyps.

④ `dealloc_bdfzinf`.
Deallokieren der dynamischen Feldkomponenten des Datentyps.

⑤ `check_valid_bdfzinf`.
Prüfen der Gültigkeit der Komponenten.

6.1.8 Datentyp `t_bdfzp`

Alle in einer BDF-Datei des Typs `RI` enthaltenen Informationen, die im Zusammenhang mit verschiedenen Zeitangaben stehen (Ausgabezeitpunkt, Differenzzeitpunkt und charakteristischer Zeitpunkt) werden jeweils getrennt voneinander in Variablen des selbstdefinierten Datentyps `t_bdfzp` abgelegt.

Moduldatei: mod_bdf_zp.f90.

Directory: $(PROGHOME)/public/datatypes/<*system*>/.

Modulname: bdfzp.

Datentyp: t_bdfzp (Datums- und Zeitangaben).

Inhalt: enthält alle in den Info-Records Nr. 002 »charakteristischer Zeitpunkt« (siehe 4.2.3 auf Seite 11), Nr. 010 »Ausgabezeitpunkt« (siehe 4.2.11 auf Seite 18) sowie Nr. 011 »Differenzzeitpunkt« (siehe 4.2.12 auf Seite 19) abgelegten Informationen.

Komponenten:

① Anzahl charakt. Zeitpunkte *oder* Anzahl Ausgabezeitpunkte *oder* Anzahl Differenzzeitpunkte.
typ = INTEGER,
nam = ANZAHL (entspricht NZP *oder* NAZ *oder* NDZ).

② Code charakt. Zeitpunkt.
typ = INTEGER, POINTER, DIMENSION(:),
nam = CIDBEZ (entspricht BCIDBEZ),
dynamisches Feld.
Hinweis: Dieses dynamische Feld wird nur für besondere Zeitpunkte ZPTYP = BZ allokiert. Zur Bedeutung siehe Beschreibung in Unterabschnitt 4.2.3 auf Seite 11.

③ Datums- und Zeitangabe.
typ = CHARACTER (LEN=29), POINTER, DIMENSION(:),
nam = DAT_F (entspricht den zuvor in den Variablen BJAHR, BMONAT, BTAG, BSTUNDE, BMINUTE, BSEKUNDE und BNANOSEKUNDE *oder* AZJAHR, AZMONAT, AZTAG, AZSTUNDE, AZMINUTE, AZSEKUNDE und AZNANOSEKUNDE *oder* DZJAHR, DZMONAT, DZTAG, DZSTUNDE, DZMINUTE, DZSEKUNDE und DZNANOSEKUNDE abgelegten Datums- und Zeitangaben),
dynamisches Feld.

④ Art des Zeitpunkts.
typ = CHARACTER (LEN=2),
nam = ZPTYP.
Damit wird angegeben, um welche Art es sich bei den charakterisierten Zeitpunkten handelt:

AZ = Ausgabezeitpunkte;
BZ = besondere (charakteristische) Zeitpunkte;
DZ = Differenzzeitpunkte.

Hinweis: Eine ausführliche Beschreibung der Bedeutung der Komponenten befindet sich in den Unterabschnitten 4.2.3 auf Seite 11 sowie 4.2.11 auf Seite 18 und 4.2.12 auf Seite 19.

Modulunterprogramme:

① init_bdfzp.
Initialisieren der Komponenten.

② `read_bdfzp_anzahl`.
Liest die Anzahl `ANZAHL` (entspricht `NZP`, `NAZ` *oder* `NDZ`) der in einer BDF-Datei des Typs `RI` enthaltenen Zeitangaben unterschiedlicher Art ein – in Abhängigkeit vom aktuellen Wert der Komponente `ZPTYP` –, um z. B. später die dynamischen Felder geeignet dimensionieren zu können.

③ `alloc_bdfzp`.
Allokieren der dynamischen Feldkomponenten des Datentyps.

④ `dealloc_bdfzp`.
Deallokieren der dynamischen Feldkomponenten des Datentyps.

⑤ `check_valid_bdfpzp`.
Prüfen der Gültigkeit der Komponenten.

6.1.9 Superdatentyp `t_bdfall`

In den vorangehenden Abschnitten 6.1.1 bis 6.1.8 wurden verschiedene Datentypen vorgestellt, in denen ausgewählte, in einer BDF-Datei des Typs `RI` abgelegte Informationen abgespeichert werden können. Alle bislang beschriebenen Datentypen sind ihrerseits Komponenten des Superdatentyps `t_bdfall` (Info-Superdatentyp). Mit dessen Hilfe ist es möglich, alle in einer BDF-Datei des Typs `RI` abgelegten Informationen in einer einzigen Fortran90-Variablen des Typs `t_bdfall` zusammenzufassen.

Moduldatei: `mod_bdf_all.f90`.

Directory: `$(PROGHOME)/public/datatypes/`<*system*>`/`.

Modulname: `bdfall`.

Datentyp: `t_bdfall` (Info-Superdatentyp).

Inhalt: alle in einer BDF-Datei des Typs `RI` enthaltenen Informationen.

Komponenten:

① physikalische Größen.
`typ = t_bdfpg`,
`nam = PG`.
Hinweis: siehe Unterabschnitt 6.1.5 auf Seite 40.

② Datums- und Zeitangaben.
`typ = t_bdfzp`,
`nam = AZP, BZP, DZP`, mit

`AZP` = Ausgabezeitpunkt(e),

`BZP` = charakteristischer Zeitpunkt(e) und

`DZP` = Differenzzeitpunkt(e).

Hinweis: siehe Unterabschnitt 6.1.8 auf Seite 44.

③ Dateibeschreibungen.
`typ = t_bdfdinf`,
`nam = IFINF, OFINF`, mit

IFINF = INP-Dateibeschreibung(en) und

OFINF = OUT-Dateibeschreibung(en).

Hinweis: siehe Unterabschnitt 6.1.4 auf Seite 39.

④ Zusatzinformationen.
`typ = t_bdfzinf`,
`nam = ZIINF`.
Hinweis: siehe Unterabschnitt 6.1.7 auf Seite 43.

⑤ Kommentare.
`typ = t_bdfco`,
`nam = COM`.
Hinweis: siehe Unterabschnitt 6.1.3 auf Seite 38.

⑥ Positionsbezeichnungen.
`typ = t_bdfbp`,
`nam = BP`.
Hinweis: siehe Unterabschnitt 6.1.2 auf Seite 37.

⑦ Vertikalstruktur.
`typ = t_bdfvs`,
`nam = VS`.
Hinweis: siehe Unterabschnitt 6.1.6 auf Seite 42.

⑧ Charakterisierung Zeitreihe.
`typ = t_bdfazr`,
`nam = AZR`.
Hinweis: siehe Unterabschnitt 6.1.1 auf Seite 36.

⑨ Integer-Komponenten.
`typ = INTEGER`,
`nam = IAKTDATA, IAKTGRPG, LDFTYP, IAKTKNO`, mit

IAKTDATA = Anzahl der Datenpunkte (siehe auch Info-Record Nr. 003 »Anzahl der Datenpunkte« in Unterabschnitt 4.2.4 auf Seite 13),

IAKTGRPG = Größe Datengruppe (siehe auch Info-Record Nr. 009 »Datengruppe« in Unterabschnitt 4.2.10 auf Seite 17),

LDFTYP = Code-Dateityp (siehe auch Info-Record Nr. 000 »Dateikopf« in Unterabschnitt 4.2.1 auf Seite 9), und

IAKTKNO = Anzahl der Datenpunkte.

⑩ Character-Komponenten.
`typ = CHARACTER (LEN=10)`,
`nam = LCIDCED`.

LCIDCED = Kürzel-Dateityp (siehe auch Info-Record Nr. 000 »Dateikopf« in Unterabschnitt 4.2.1 auf Seite 9).

Modulunterprogramme:

① `init_bdf`.
Initialisieren der Komponenten.

② `read_bdf_anzahl`.
Ermittelt die Größe aller dynamisch zu dimensionierenden Feldkomponenten.

③ `alloc_bdf`.
Allokieren der dynamischen Feldkomponenten des Datentyps.

④ `dealloc_bdf`.
Deallokieren der dynamischen Feldkomponenten des Datentyps.

⑤ `check_valid_bdf`.
Prüfen der Gültigkeit der Komponenten.

6.2 Datentypen für BDF-Daten

6.2.1 Datentypen für Hilfsfelder

Beim Lesen und Schreiben eines BDF-Datenrecords werden verschiedene Hilfsfelder benötigt, die in verschiedenen selbstdefinierten Datentypen zusammengefaßt werden.

Moduldatei: `mod_BdfData.f90`.

Directory: `$(PROGHOME)/public/datatypes/`<*system*>`/`.

Modulname: `mod_BdfData`.

Datentyp: `t_bdf_ld_i2` (Hilfsfeld `I2FELD`).

Komponenten:

① Speicherbereich `I2FELD`.
`typ = INTEGER`,
`nam = LEN`.
Aktuell allokierte Länge des Hilfsfeldes `I2FELD`, mit `LEN` $\geq$ `1`.

② Hilfsfeld `I2FELD`.
`typ = INTEGER*2, POINTER, DIMENSION(:)`,
`nam = DATA` (entspricht `I2FELD`),
dynamisches Feld.

Datentyp: `t_bdf_ld_i` (Hilfsfeld `ILOCAL`).

Komponenten:

① Speicherbereich `ILOCAL`.
`typ = INTEGER`,
`nam = LEN`.
Aktuell allokierte Länge des Hilfsfeldes `ILOCAL`, mit `LEN` $\geq$ `1`.

② Hilfsfeld `ILOCAL`.
`typ = INTEGER, POINTER, DIMENSION(:)`,
`nam = DATA` (entspricht `ILOCAL`),
dynamisches Feld.

Datentyp: `t_bdf_ld_r` (Hilfsfeld `RLOCAL`).

Komponenten:

① Speicherbereich RLOCAL.
typ = INTEGER,
nam = LEN.
Aktuell allokierte Länge des Hilfsfeldes RLOCAL, mit LEN $\geq$ 1.

② Hilfsfeld RLOCAL.
typ = REAL, POINTER, DIMENSION(:),
nam = DATA (entspricht RLOCAL),
dynamisches Feld.

Datentyp: t_bdf_ld_x (Hilfsfeld CLOCAL).

Komponenten:

① Speicherbereich CLOCAL.
typ = INTEGER,
nam = LEN.
Aktuell allokierte Länge des Hilfsfeldes CLOCAL, mit LEN $\geq$ 1.

② Hilfsfeld CLOCAL.
typ = COMPLEX, POINTER, DIMENSION(:),
nam = DATA (entspricht CLOCAL),
dynamisches Feld.

Datentyp: t_bdf_ld_d (Hilfsfeld DLOCAL).

Komponenten:

① Speicherbereich DLOCAL.
typ = INTEGER,
nam = LEN.
Aktuell allokierte Länge des Hilfsfeldes DLOCAL, mit LEN $\geq$ 1.

② Hilfsfeld DLOCAL.
typ = DOUBLE PRECISION, POINTER, DIMENSION(:),
nam = DATA (entspricht DLOCAL),
dynamisches Feld.

Hinweis: Weitere Informationen zur Dimensionierung der Felder sind auch in dem Abschnitt 5.3 auf Seite 30 enthalten.

6.2.2 Datentypen für Datenfelder

In Abhängigkeit von dem FORTRAN-Datentyp werden BDF-Daten in verschiedenen Feldern abgelegt. Hierfür stehen die nachfolgend angegebenen selbstdefinierten Datentypen zur Verfügung.

Moduldatei: mod_BdfData.f90.

Directory: $(PROGHOME)/public/datatypes/<*system*>/.

Modulname: mod_BdfData.

Datentyp: t_bdf_gd_i (Datenfeld IGLOBAL).

Komponenten:

① Speicherbereich IGLOBAL.
typ = INTEGER, DIMENSION(3),
nam = LEN.
Aktuell allokierte Länge des Feldes IGLOBAL, wobei für jede Komponente i LEN(i) $\geq$ 1 gilt.

② Datenfeld IGLOBAL.
typ = INTEGER, POINTER, DIMENSION(:,:,:),
nam = DATA (entspricht IGLOBAL),
dynamisches Feld.

Datentyp: t_bdf_gd_r (Datenfeld RGLOBAL).

Komponenten:

① Speicherbereich RGLOBAL.
typ = INTEGER, DIMENSION(3),
nam = LEN.
Aktuell allokierte Länge des Feldes RGLOBAL, wobei für jede Komponente i LEN(i) $\geq$ 1 gilt.

② Datenfeld RGLOBAL.
typ = REAL, POINTER, DIMENSION(:,:,:),
nam = DATA (entspricht RGLOBAL),
dynamisches Feld.

Datentyp: t_bdf_gd_x (Datenfeld CGLOBAL).

Komponenten:

① Speicherbereich CGLOBAL.
typ = INTEGER, DIMENSION(3),
nam = LEN.
Aktuell allokierte Länge des Feldes CGLOBAL, wobei für jede Komponente i LEN(i) $\geq$ 1 gilt.

② Datenfeld CGLOBAL.
typ = COMPLEX, POINTER, DIMENSION(:,:,:),
nam = DATA (entspricht CGLOBAL),
dynamisches Feld.

Datentyp: t_bdf_gd_d (Datenfeld DGLOBAL).

Komponenten:

① Speicherbereich DGLOBAL.
typ = INTEGER, DIMENSION(3),
nam = LEN.
Aktuell allokierte Länge des Feldes DGLOBAL, wobei für jede Komponente i LEN(i) $\geq$ 1 gilt.

② Datenfeld DGLOBAL.
typ = DOUBLE PRECISION, POINTER, DIMENSION(:,:,:),
nam = DATA (entspricht DGLOBAL),
dynamisches Feld.

Hinweis: Weitere Informationen zur Dimensionierung der Felder sind auch in dem Abschnitt 5.3 auf Seite 30 enthalten.

6.2.3 Superdatentyp t_bdf_data

Das Lesen und Schreiben der BDF-Daten von bzw. in eine Datei des Typs DD wird durch Verwenden des Superdatentyps t_bdf_data (Data-Superdatentyp) erleichtert, da alle hierfür benötigten Informationen und Daten in einer einzigen Fortran90-Variablen des Typs t_bdf_data zusammengefaßt werden können.

Moduldatei: mod_BdfData.f90.

Directory: $(PROGHOME)/public/datatypes/<*system*>/.

Modulname: mod_BdfData.

Datentyp: t_bdf_data (Data-Superdatentyp).

Inhalt: alle zum Lesen / Schreiben von BDF-Daten aus / in eine Datei des Typs DD benötigten Variablen und Felder.

Komponenten der Hilfs- und Datenfelder:

① Hilfsfeld I2FELD.
typ = t_bdf_ld_i2,
nam = I2FELD.

② Hilfsfeld ILOCAL.
typ = t_bdf_ld_i,
nam = ILOCAL.

③ Hilfsfeld RLOCAL.
typ = t_bdf_ld_r,
nam = RLOCAL.

④ Hilfsfeld CLOCAL.
typ = t_bdf_ld_x,
nam = CLOCAL.

⑤ Hilfsfeld DLOCAL.
typ = t_bdf_ld_d,
nam = DLOCAL.

⑥ Datenfeld IGLOBAL.
typ = t_bdf_gd_i,
nam = IGLOBAL.

⑦ Datenfeld RGLOBAL.
typ = t_bdf_gd_r,
nam = RGLOBAL.

⑧ Datenfeld CGLOBAL.
typ = t_bdf_gd_x,
nam = CGLOBAL.

⑨ Datenfeld DGLOBAL.
typ = t_bdf_gd_d,
nam = DGLOBAL.

Komponenten des Typs INTEGER:

① Code-Nummer.
typ = INTEGER,
nam = ICODE.
Code-Nummer der zu lesenden / schreibenden physikalischen Größe. Die Code-Nummer muß dabei den in den Konfigurationsdateien festgelegten Werten entsprechen (siehe Abschnitt 3.2 auf Seite 4).

② Anzahl der Datenpunkte.
typ = INTEGER,
nam = IAKTDATA.
Anzahl der Datenpunkte des Datenrecords. Siehe hierzu auch die Unterabschnitte 4.2.4 auf Seite 13 sowie 6.1.9 auf Seite 46.

③ Dimension physik. Größe.
typ = INTEGER,
nam = NDIM.
Dimension der physikalischen Größe. Siehe hierzu auch die Unterabschnitte 4.2.14 auf Seite 21 sowie 6.1.5 auf Seite 40.

④ Anzahl Datenvariationen physik. Größe.
typ = INTEGER,
nam = NVAR.
Anzahl der Datenvariationen der physikalischen Größe. Siehe hierzu auch die Unterabschnitte 4.2.19 auf Seite 24 sowie 6.1.5 auf Seite 40.

⑤ Dimensionsverschiebung.
typ = INTEGER,
nam = IADD.
Die Unterprogramme dadowr.f und dadorr.f lassen die Verschiebung einzelner Komponenten der Daten beim Lesen oder Schreiben zu. Von dieser Möglichkeit wurde bislang kein Gebrauch gemacht, und es ist auch fraglich, ob davon jemals Gebrach gemacht werden wird.

⑥ Recordnummer.
typ = INTEGER,

nam = IREC.
Bei Berechnung der Recordnummer durch das Modulunterprogramm eval_irec wird diese hier eingetragen.

Komponenten des Typs REAL:

① Datenminimum.
typ = REAL,
nam = RLOW.

② Datenmaximum.
typ = REAL,
nam = RHIGH.

③ Datenintervall.
typ = REAL,
nam = RDIF.

④ Diskretisierungsfehler.
typ = REAL,
nam = RERRMAX.

⑤ unterer Ersetzungswert.
typ = REAL,
nam = XRMIN.

⑥ oberer Ersetzungswert.
typ = REAL,
nam = XRMIN.

Hinweis: Angaben zu Datenmaximum, Datenminimum, Datenintervall und Diskretisierungsfehler sowie zu oberer Ersetzungswert und unterer Ersetzungswert werden direkt mit dem einzelnen Daten-Record in die BDF-Datei des Typs DD geschrieben. Vor dem Schreiben müssen RERRMAX und ggf. auch XRMIN und XRMAX bekannt sein, falls die Daten in den Formen RI2 oder RR2 abgespeichert werden sollen (siehe hierzu auch Abschnitt 4.2.15 auf Seite 21).

Komponenten des Typs CHARACTER:

① Speicherformat.
typ = CHARACTER (LEN=6),
nam = STORAGE, bezeichnet das Speicherformat der Daten. Folgende Werte sind zulässig:

VECTOR: Daten sind im Speicherformat >VECTOR< abgelegt. Die *zweite* Komponente der Datenfelder läuft über die Dimension der physik. Größen.

TICAD: Daten sind im Speicherformat >TICAD< abgelegt. Die *erste* Komponente der Datenfelder läuft über die Dimension der physik. Größen.

Hinweis: Weitere Informationen zum Speicherformat in Abschnitt 5.3 auf Seite 30.

② Art physik. Größe.
typ = CHARACTER (LEN=6),
nam = DATENART.

Hinweis: Weitere Informationen zur Art der physik. Größen sind in den Unterabschnitten 4.2.13 auf Seite 20 und 6.1.5 auf Seite 40 zu finden.

③ FOTRAN-Datentyp.
`typ = CHARACTER (LEN=3)`,
`nam = DATENTYP`.
Hinweis: Weitere Informationen zum FORTRAN-Datentyp sind in den Unterabschnitten 4.2.15 auf Seite 21 und 6.1.5 auf Seite 40 zu finden.

④ Typ der Daten.
`typ = CHARACTER (LEN=20)`,
`nam = TYPE_OF_DATA`. Folgende Werte sind zulässig:

`ZEITREIHE`: Daten sind vom Typ >äquidistante Zeitreihe<.

`SYNOPTISCH`: Daten sind vom Typ >synoptischer Datensatz<.

`EINZELANALYSE`: Daten sind vom Typ >Ergebnisse Einzelanalyse<.

`DIFFERENZANALYSE`: Daten sind vom Typ >Ergebnisse Differenzanalyse<.

Hinweis: Weitere Informationen zum Typ der Daten in Abschnitt 5.3 auf Seite 30.

Modulunterprogramme:

① `init_bdf_data`.
Initialisieren der Komponenten. Auch zum Deallokieren dynamischer Feldkomponenten.

② `config_bdf_data`.
Konfigurieren aller Komponenten einer Variablen des Datentyps, mit Ausnahme der Recordnummer.

③ `alloc_bdf_data`.
Allokieren der dynamischen Feldkomponenten des Datentyps.

④ `generate_bdf_data`.
Zusammenfassen aller Aktionen der Modulunterprogramme `init_bdf_data`, `config_bdf_data` und `alloc_bdf_data`.

⑤ `eval_irec`.
Berechnung der Recordnummer und speichern in einer Variablen des Typs `t_bdf_data`.

⑥ `read_bdf_data`.
Lesen eines Daten-Records.

⑦ `write_bdf_data`.
Schreiben eines Daten-Records.

7 Benutzen der Fortran90-Schnittstellen

In diesem Kapitel wird aufgezeigt, wie die unter Fortran90 vorhandenen selbstdefinierten Datentypen und Modulunterprogramme den Umgang mit BDF-Informationen und -Daten im Vergleich mit den konventionellen FORTRAN77-Schnittstellen vereinfachen und sicherer machen. Dieses wurde erreicht durch die

1. Nutzung abgeleiteter Datentypen zur Kapselung von zusammenhängenden Daten, die in Komponenten des Datentyps abgelegt sind;
2. Bereitstellen von Elementarfunktionen für einen Datentyp; des weiteren durch
3. dynamische Speicherplatzreservierung für Komponenten eines abgeleiteten Datentyps mit `POINTER`-Attribut, wodurch in Unterprogrammen ausreichend Speicherplatz zur Aufnahme von Daten in der Variablen eines abgeleiteten Datentyps vereinbart werden kann.

Selbstdefinierte Datentypen, dynamische Speicherplatzreservierung und bereitgestellte Elementarfunktionen wurden in dem vorangehenden Kapitel 6 ausführlich vorgestellt.

7.1 Initialisierungsphase

Die Initialisierungsphase muß in einem Anwendungsprogramm, welches BDF-Daten verarbeiten möchte allen anderen Aktionen vorausgehen und unmittelbar nach dem Start des Programmes erfolgen. In dem Abschnitt 5.1 auf Seite 27 wurden die in FORTRAN77-Programmen durchzuführenden Schritte erläutert. Bei neu zu erstellenden Fortran90-Programmen sollte das Hauptprogramm aus der Musterdatei `hp_muster.f90` abgewandelt werden (für Fortran90-Module ist die Musterdatei `up_muster.f90` vorhanden). Die Musterdateien sind in dem Verzeichnis `$(PROGHOME)/mus/` abgelegt. In den Musterdateien werden Elemente des Moduls `baw_program` aus der Moduldatei `mod_baw_program.f90` (Public-Verzeichnis `datatypes`) verfügbar gemacht. Unter Beachtung der vorgenannten Voraussetzungen ist zur Initialisierung eines Fortran90-Programmes, das mit BDF-Daten arbeiten soll, zunächst das Modulunterprogramm `baw_program_start`

```
CALL baw_program_start &
     (p_var, do_com=.true.)
```

auszuführen, welches die immer wiederkehrenden Initialisierungsschritte eines Fortran90-Programms durchläuft. Die Angabe des optionalen Parameters `do_com` ist dabei unbedingt erforderlich, da nur dann das Unterprogramm `chocom.f` durchlaufen wird; weitere optionale Parameter des Modulunterprogramms können bei Bedarf ergänzend angegeben werden. Danach müssen noch die Unterprogramme

1. `chospr.f` (optional),
2. `dirini.f` (unbedingt erforderlich),
3. `phydef.f` (unbedingt erforderlich) und
4. `dirdef.f` (unbedingt erforderlich)

ausgeführt werden. Ausführliche Angaben zu den Funktionen der vorgenannten Unterprogramme sind in dem Abschnitt 5.1 auf Seite 27 enthalten. In dem Anwendungsprogramm DIDAMINTQ wurde die Initialisierungsphase beispielsweise in der Form

```
CALL baw_program_start &
     (p_var, do_com=.true.)
CALL dirini
CALL phydef
CALL dirdef
```

realisiert.

7.2 Deklaration der BDF-Variablen

Die Deklaration der BDF-Variablen geht dem Verwenden der unter Fortran90 neu geschaffenen Schnittstellen voraus. Es werden minimal zwei Variablen benötigt, wovon `bdf_info` (Info-Superdatentyp) der Aufnahme der BDF-Informationen und `bdf_data` (Data-Superdatentyp) dem Speichern von BDF-Daten dient. Der Deklaration der vorgenannten Variablen, die als einfache Variablen oder als (allokierbare) Felder definiert werden können, müssen die Use-Anweisungen

```
USE bdfall,            ONLY : t_bdfall
USE mod_BdfData,       ONLY : t_bdf_data
```

vorausgehen. Dadurch werden aus den Modulen `bdfall` (Moduldatei `mod_bdf_all.f90` im Public-Verzeichnis `datatypes`) und `mod_BdfData` (Moduldatei `mod_BdfData.f90` im Public-Verzeichnis `datatypes`) die Datentypen `t_bdfall` und `t_bdf_data` der jeweiligen Programmeinheit verfügbar gemacht. Im Anschluß daran können die BDF-Variablen als einfache Variable gemäß

```
TYPE (t_bdfall)            :: bdf_info
TYPE (t_bdf_data)          :: bdf_data
```

oder, falls diese als Felder benötigt werden, z. B. mit Hilfe der Deklaration

```
TYPE (t_bdfall)   , ALLOCATABLE, DIMENSION(:) :: bdf_info
TYPE (t_bdf_data) , ALLOCATABLE, DIMENSION(:) :: bdf_data
```

vereinbart werden. Den allokierbaren Feldern muß anschließend noch durch `allocate` Speicher in geeigneter Größe zugewiesen werden. Jede in dieser Weise deklarierte Variable oder Feldkomponente kann später entweder den Inhalt einer BDF-Datei des Typs `RI` (Variable `bdf_info`) oder einen Daten-Record einer BDF-Datei des Typs `DD` aufnehmen (Variable `bdf_data`).

7.3 Info-Records Lesen und Verarbeiten

Beim Arbeiten mit BDF-Info-Variablen der Art `bdf_info` (Info-Superdatentyp) zur Aufnahme von Informationen aus einer BDF-Datei des Typs `RI` müssen stets die Schritte

1. Initialisieren der Komponenten,
2. Ermitteln der Größe aller dynamisch zu dimensionierenden Feldkomponenten,
3. Allokieren der dynamischen Feldkomponenten,
4. Lesen der BDF-Infodaten und
5. Deallokieren der dynamischen Feldkomponenten

in der angegebenen Reihenfolge durchlaufen werden.

7.3.1 Lesen in eine BDF-Info-Variable

Beim Lesen in eine Info-Variable werden die in einer BDF-Datei des Typs `RI` abgespeicherten Informationen zunächst in den dafür vorgesehenen Feldern und Variablen der Include-Datei `direkt.h` abgespeichert. Anschließend werden diese Daten in eine Variable des Typs `bdf_info` übertragen. Bei diesem Vorgang werden die vorgenannten Schritte 1 bis 4 durchlaufen. Vorausgesetzt das Programm wurde korrekt initialisiert und die Variable `bdf_info` ordentlich deklariert, dann muß noch mit Hilfe einer Use-Anweisung das in dem Modul `mod_bdf_info_io` befindliche Modulunterprogramm `BdfInfoRead` (aus der Moduldatei `mod_bdf_info_io.f90` im Public-Verzeichnis `record`) der aktuellen Programmeinheit verfügbar gemacht werden. Die vorgenannten Schritte 1 bis 4 werden durch Aufruf von

```
CALL BdfInfoRead &
     (tracefile, printfile, infofile, recfile, datfile, &
     laktdf, bdf_info, fehler)
```

in einem einzigen Unterprogramm abgearbeitet. Jetzt stehen sämtliche Informationen in `bdf_info` der weitergehenden Bearbeitung zur Verfügung.

7.3.2 Auswerten einer BDF-Info-Variablen

Zuallererst soll an dieser Stelle an die unter FORTRAN77 vorhandenen Möglichkeiten beim Auswerten der Info-Records erinnert werden (siehe hierzu den Unterabschnitt 5.2.2 auf Seite 29). Diese können auch unter Fortran90 benutzt werden. Für das Auswerten einer BDF-Info-Variablen existieren derzeit folgende Fortran90-Moduldateien mit jeweils mehreren Modulunterprogrammen:

`mod_check_bdfinfo.f90`: Diese Moduldatei aus dem Public-Verzeichnis `record` enthält den Modul `mod_check_bdfinfo` mit den Modulunterprogrammen

- **`check_zeitraum`** : falls es sich um Daten des Typs >synoptischer Datensatz< handelt wird u. a. geprüft, ob ein gewünschter Zeitraum in den Daten enthalten ist oder nicht;
- **`check_zeitraum_azr`** : falls es sich um Daten des Typs >äquidistante Zeitreihe< handelt wird u. a. geprüft, ob ein gewünschter Zeitraum in den Daten enthalten ist oder nicht;
- **`equal_bdf_zp`** : falls es sich um Daten des Typs >synoptischer Datensatz< handelt wird geprüft, ob in unterschiedlichen BDF-Dateien abgelegte Daten für dieselben Termine (Ausgabezeitpunkte) vorhanden sind oder nicht;
- **`equal_bdf_azr`** : falls es sich um Daten des Typs >äquidistante Zeitreihe< handelt wird geprüft, ob in unterschiedlichen BDF-Dateien abgelegte Daten für dieselben Zeiträume (Charakterisierung Zeitreihe) vorhanden sind oder nicht;
- **`equal_bdf_3dvs`** : prüft, ob die Vertikalstrukturen der in unterschiedlichen BDF-Dateien abgelegten Daten identisch sind oder nicht;
- **`equal_bdf_pg`** : prüft, ob in unterschiedlichen BDF-Dateien eine physikalische Größe mehrfach vorhanden ist oder nicht;
- **`equal_bdf_typ`** : prüft, ob der Typ der Daten zwischen unterschiedlichen BDF-Dateien identisch ist oder nicht.

equal_bdf_system : prüft, ob die zu unterschiedlichen BDF-Dateien gehörenden Systemdateien identisch sind oder nicht (Code-Dateityp, Typ der Systemdatei).

mod_bdf_zinf_jobs.f90: Diese Moduldatei aus dem Public-Verzeichnis `record` enthält den Modul `mod_bdf_zinf_jobs` mit den Modulunterprogrammen

extract_zusatzinfo : extrahieren von Zusatzinformationen aus Zeichenketten (Klartext Zusatzinformation);

generate_zinf_data : aus Zusatzinformationen unterschiedlicher BDF-Dateien wird ein neues Objekt des Datentyps `t_bdfzinf` erzeugt, in das jede vorhandene Information genau einmal aufgenommen und ggf. noch eine weitere zur Aufnahne der Grenzwassertiefe `HLIMIT` hinzugefügt wird.

mod_lib_record.f90: Diese Moduldatei aus dem Public-Verzeichnis `record` enthält den Modul `lib_record` mit den zum Auswerten von BDF-Info-Variablen nützlichen Modulunterprogrammen

m_gen_dad_typ : ermittelt aus einem Objekt des Typs `t_bdfall` den Typ der Daten unter Verwendung des externen Unterprogramms `gen_dad_typ.f`;

m_getsys : ermittelt aus einem Objekt des Typs `t_bdfall` den Typ der Systemdatei.

m_xzi : ermittelt aus einem Objekt des Typs `t_bdfall` Zusatzinformationen für die Grenzwassertiefe `HLIMIT`, den Steuerparameter für die morphodyn. Topographie `IMORPHO`, die konstante Dichte `DENSITY`, die Schwerebeschleunigung `GRAVITY`, den Wert konstanter Luftdruck `P0`, den Wert konstanter Salzgehalt `SALT` sowie den Wert konstante Temperatur `TEMP`.

m_erm_struktur : ermittelt aus einem Objekt des Typs `t_bdfall`, ob eine Vertikalstruktur zwei- oder dreidimensional ist, unter Verwendung des externen Unterprogramms `erm_struktur.f`;

m_dadspq : ermittelt aus einem Objekt des Typs `t_bdfall`, ob Informationen über eine physikalische Größe mit der Code-Nummer `ICODE` in dem Objekt vorhanden sind oder nicht, wobei auf das externe Unterprogramm `dadspq.f` zurückgegriffen wird.

m_ermrwf : ermittelt ob für eine physikalische Größe mit der Code-Nummer `ICODE` eine Referenzwasserfläche existiert oder nicht, wobei auf das externe Unterprogramm `getrwf.f` zurückgegriffen wird.

mod_lib_trim.f90: Diese Moduldatei aus dem Public-Verzeichnis `trim` enthält den Modul `lib_trim` mit den zum Auswerten von BDF-Info-Variablen nützlichen Modulunterprogrammen

m_chkh : prüft für zwei Objekte vom Typ `t_bdfall`, ob die Vertikalstrukturen übereinstimmen oder nicht; hierbei wird das externe Unterprogramm `chkh.f` benutzt.

7.3.3 Explizite Übernahme der Info-Records

Das in Unterabschnitt 7.3.1 auf Seite 57 geschilderte Vorgehen bei der Übernahme der Info-Records stellt den bequemsten und sichersten Weg dar. Da es keinen Grund gibt davon abzuweichen sollte man es auch nicht tun.

7.3.4 Deallokieren einer BDF-Info-Variablen

Vor dem Deallokieren einer BDF-Info-Variablen muß mit Hilfe einer Use-Anweisung das in dem Modul `bdfall` befindliche Modulunterprogramm `dealloc_bdf` (aus der Moduldatei `mod_bdf_all.f90` im Public-Verzeichnis `datatypes`) der aktuellen Programmeinheit verfügbar gemacht werden. Danach kann mittels

```
CALL dealloc_bdf &
     (ltest, tracefile, bdf_info, fehler)
```

jener der Variablen `bdf_info` zuvor dynamisch zugewiesene Speicherbereich deallokiert werden.

7.4 Speicherung der Daten

Bezüglich allgemeiner Informationen zum Thema Speicherung der Daten wird auf die im Zusammenhang mit der Beschreibung der FORTRAN77-Schnittstelle gegebenen Erläuterungen in dem Abschnitt 5.3 auf Seite 30 verwiesen.

7.4.1 Dimensionierung

An die korrekte Dimensionierung der zur Aufnahme von BDF-Daten benötigten Felder braucht der Anwendungsprogrammierer bei Verwenden der Fortran90-Schnittstelle keine großen Gedanken mehr zu verschwenden. Unter der Voraussetzung, daß die in den Abschnitten 7.1 bis 7.3 genannten Schritte durchgeführt wurden, muß noch durch eine Use-Anweisung das in dem Modul `mod_BdfData` (Moduldatei `mod_BdfData.f90` im Public-Verzeichnis `datatypes`) vorhandene Modulunterprogramm `generate_bdf_data` der aktuellen Programmeinheit verfügbar gemacht werden. Da dann sowohl die BDF-Variable `bdf_info` (Info-Superdatentyp) wie auch die Include-Datei `direkt.h` korrekte BDF-Informationen enthält, werden durch

```
CALL generate_bdf_data &
     (tracefile, printfile, bdf_data, icode, laktdf, &
      bdf_info, fehler)
```

alle zur Aufnahme von Daten benötigten dynamischen Feldkomponenten in einer BDF-Variablen `bdf_data` (Data-Superdatentyp) in Abhängigkeit von der Code-Nummer `ICODE` korrekt allokiert. Nach dem Allokieren einer BDF-Data-Variablen werden die Daten dann z. B. beim späteren Lesen aus einer BDF-Datei des Typs `DD` in einer der zum Datentyp `t_bdf_data` gehörenden Feldkomponenten (`IGLOBAL`, `RGLOBAL`, `CGLOBAL` oder `DGLOBAL`) abgespeichert (Subkomponente Datenfeld `DATA`). Die aktuelle Größe des Datenfeldes ist in der jeweiligen Subkomponente Speicherbereich `LEN` abgelegt.

7.4.2 Deallokieren einer BDF-Data-Variablen

Vor dem Deallokieren einer BDF-Data-Variablen muß via Use-Anweisung das in dem Modul mod_BdfData befindliche Modulunterprogramm init_bdf_data (aus der Moduldatei mod_BdfData.f90 im Public-Verzeichnis datatypes) der aktuellen Programmeinheit verfügbar gemacht werden. Danach kann mittels

```
CALL init_bdf_data &
     (tracefile, printfile, bdf_data, fehler, TYP='DAL')
```

jener der Variablen bdf_data zuvor dynamisch zugewiesene Speicherbereich deallokiert werden.

7.5 Daten-Records Lesen

Auch bei Verwenden der Fortran90-BDF-Schnittstelle zerfällt das Lesen des in einer BDF-Datei vom Typ DD abgelegten Datensatzes in die Teilschritte Datei öffnen, Recordnummer berechnen, Daten-Record lesen und Datei schließen. Werden mehrere Daten-Records hintereinander aus derselben Datei gelesen werden, so ist es natürlich zweckmäßig, daß diese nur einmal geöffnet und auch nur einmal geschlossen wird. Für die nachfolgend beschriebenen Schritte wird vorausgesetzt, daß die in den vorangehenden Abschnitten beschriebenen Schritte der Initialisierung, Deklaration, Lesen in eine BDF-Info-Variable des Typs t_bdfall sowie Dimensionierung der Variablen des Typs t_bdf_data korrekt durchlaufen wurden.

7.5.1 Daten-Datei Öffnen

Für das Öffnen einer Daten-Datei steht das in dem Modul lib_record (Moduldatei mod_lib_record.f90 aus dem Public-Verzeichnis record) vorhandene Modulunterprogramm m_dadodf zur Verfügung. Dieses muß via Use-Anweisung für die lokale Programmeinheit nutzbar gemacht werden. Durch Ausführen von

```
CALL m_dadodf &
     (p_var, datfile)
```

wird die Datei datfile mit Hilfe des Modulunterprogramms universal_openfile geöffnet (aus Modul baw_files in der Moduldatei mod_baw_files.f90 aus dem Public-Verzeichnis datatypes). Werden die optionalen Parameter LAKTDF und LAKTOF angegeben, so wird das Öffnen der Datei mit dem externen Unterprogramm dadodf.f durchgeführt.

7.5.2 Daten-Datei Schließen

Für das Schließen einer Daten-Datei steht das in dem Modul lib_record (Moduldatei mod_lib_record.f90 aus dem Public-Verzeichnis record) vorhandene Modulunterprogramm m_dadcdf zur Verfügung. Dieses Modulunterprogramm muß mit Hilfe einer Use-Anweisung der lokalen Programmeinheit nutzbar gemacht werden. werden. Durch Ausführen von

```
CALL m_dadcdf &
     (p_var, datfile)
```

wird die Datei datfile über das Modulunterprogramm universal_closefile geschlossen (aus Modul baw_files in der Moduldatei mod_baw_files.f90 aus dem Public-Verzeichnis datatypes). Werden die optionalen Parameter LAKTDF und LAKTOF angegeben, so wird das Schließen der Datei mit dem externen Unterprogramm dadcdf.f durchgeführt.

Es ist darauf zu achten, daß Öffnen und Schließen von BDF-Dateien des Typs DD symmetrisch, mit identischen Parameterlisten durchzuführen.

7.5.3 Recordnummer berechnen

Wurden die in dem Unterabschnitt 7.4.1 auf Seite 59 genannten Schritte absolviert, so muß vor dem Lesen des Datensatzes noch die Berechnung der Recordnummer erfolgen. Hierzu ist das in dem Modul mod_BdfData befindliche Modulunterprogramm eval_irec der aktuellen Programmeinheit mit einer Use-Anweisung verfügbar zu machen (aus der Moduldatei mod_BdfData.f90 in dem Public-Verzeichnis datatypes). Das Berechnen der Recordnummer erfolgt durch

```
CALL eval_irec &
     (tracefile, printfile, icode, iaktkno, acttimestep, laktdf, &
      bdf_data, fehler)
```

und bewirkt, daß in der Komponente IREC (Recordnummer) der Variablen bdf_data vom Typ t_bdf_data (Data-Superdatentyp) die Recordnummer für den zu lesenden Datensatz eingespeichert wird. Das Ergebnis der Berechnung, die mit dem externen Unterprogramm dadcr.f durchgeführt wird, ist abhängig von der Code-Nummer der physikalischen Größe ICODE, der Nummer des Knotens (Gitterpunktes) IAKTKNO (falls die Datensatzart >äquidistante Zeitreihe< gelesen werden soll) *oder* der Nummer ACTTIMESTEP des Ausgabezeitpunktes (falls die Datensatzarten >synoptischer Datensatz< *oder* >Ergebnisse Einzelanalyse< *oder* >Ergebnisse Differenzanalyse< gelesen werden sollen).

7.5.4 Record lesen

Unter der Voraussetzung, daß die Berechnung der Recordnummer IREC zuvor korrekt erfolgte (siehe vorangehenden Abschnitt), kann das Lesen eines Daten-Records mit Hilfe des Modulunterprogrammes read_bdf_data erfolgen, falls der Modul mod_BdfData zuvor mit einer Use-Anweisung der lokalen Programmeinheit bekanntgemacht wurde (Moduldatei mod_BdfData.f90 in dem Public-Verzeichnis datatypes). Das Lesen der Daten geschieht mit

```
CALL read_bdf_data &
     (tracefile, printfile, datfile, bdf_data, fehler)
```

aus der Datei datfile. Dabei nutzt das Modulunterprogramm das externe Unterprogramm dadorr.f. Danach stehen die Daten in einer Komponente der Variablen bdf_data zur weiteren Verarbeitung bereit (siehe hierzu gegebenenfalls nochmals den Unterabschnitt 7.4.1 auf Seite 59).

7.6 Schreiben aus einer BDF-Info-Variablen

In diesem Abschnitt wird das Schreiben von Info-Records aus einer BDF-Info-Variablen für Standardanwendungen unter Fortran90 kurz beschrieben (für FORTRAN77 siehe Abschnitt 5.5 auf Seite 34). In Anwendungsprogrammen, die selbst neue BDF-Dateien erstellen, muß vor dem Schreiben der ersten Daten-Records die zu der BDF-Datei des Typs `DD` korrespondierende BDF-Datei des Typs `RI` erstellt worden sein. Grundsätzlich müssen vor dem Erzeugen und Beschreiben einer Datei des Typs `RI` alle über die abzuspeichernden Daten benötigten Informationen in den Variablen und Feldern der Include-Datei `direkt.h` korrekt abgelegt worden sein.

7.6.1 Übertragen und Schreiben der Info-Records

Für das Übertragen der Info-Records unter Fortran90 müssen zunächst alle Informationen in einer BDF-Info-Variablen `bdf_info` (Info-Superdatentyp `t_bdfall`) gesammelt und danach in die Felder und Variablen der Include-Datei `direkt.h` übertragen werden. Hierbei sind mehrere Schritte zu durchlaufen.

1. Initialisieren der Komponenten: Bei diesem Schritt ist mit Hilfe des in dem Modul `bdfall` vorhandenen Modulunterprogramms `init_bdf` (aus der Moduldatei `mod_bdf_all.f90` im Public-Verzeichnis `datatypes`) durch Aufruf von

   ```
   CALL init_bdf &
        (ltest, tracefile, bdf_info, fehler)
   ```

   die Variable `bdf_info` mit voreingestellten Werten zu initialisieren.

2. Ermitteln der Größe der Feldkomponenten: Wie in Abschnitt 6.1.9 auf Seite 46 beschrieben, enthält eine Variable des Typs `t_bdfall` mehrere dynamische Komponenten, deren Größe vor dem Allokieren bestimmt werden muß. Dies ist der Fall für die Komponenten

   (a) `t_bdfpg` (physikalische Größen),
   (b) `t_bdfzp` (Datums- und Zeitangaben),
   (c) `t_bdfdinf` (Dateibeschreibungen),
   (d) `t_bdfzinf` (Zusatzinformationen)
   (e) `t_bdfco` (Kommentare),
   (f) `t_bdfbp` (Positionsbezeichnungen),
   (g) `t_bdfvs` (Vertikalstruktur) und
   (h) `t_bdfazr` (Charakterisierung Zeitreihe).

   Für alle sind die Subkomponente `ANZAHL` sowie einige weitere statische Subkomponenten geeignet zu belegen.

   Neben den dynamisch allokierbaren Feldkomponenten einer Variablen des Datentyps `t_bdfall` müssen ferner auch den Integer-Komponenten `IAKTDATA`, `IAKTGRPG`, `LDFTYP`, `IAKTKNO` sowie den Character-Komponenten `LCIDCED` geeignete Werte zugewiesen werden.

3. Allokieren der Feldkomponenten: Nach dem Initialisieren und Belegen der statischen Feldkomponenten der Variablen des Typs `t_bdfall` kann den dynamischen Komponenten der Variablen mit Hilfe des in dem Modul `bdfall` befindlichen Modulunterprogramms `alloc_bdf` (aus der Moduldatei `mod_bdf_all.f90` im Public-Verzeichnis `datatypes`) durch Aufruf von

   ```
   CALL alloc_bdf &
        (ltest, tracefile, bdf_info, fehler)
   ```

   Speicherplatz zugewiesen werden. Alternativ ist dies auch komponentenweise mit den spezifischen Modulunterprogrammen `alloc_bdfpg`, `alloc_bdfzp`, `alloc_bdfdinf`, `alloc_bdfzinf`, `alloc_bdfco`, `alloc_bdfbp` und `alloc_bdfvs` möglich.

4. Eintragen der Information: Nach dem Allokieren der dynamischen Komponenten sind diesen alle weiteren erforderlichen Angaben zuzuweisen.

Als Beispiel zu den vorgenannten Schritten soll an dieser Stelle auf das zu dem Anwendungsprogramm DIDAMINTQ gehörende Modul `DidaMintQ_bdf_mkinfo` (Moduldatei `mod_DidaMintQ_bdf_mkinfo.f90`) hingewiesen werden. Dort steuert das Modulunterprogramm `infoobject` den oben beschriebenen Ablauf.

Übertragen der Informationen in die Includedatei `direkt.h` und Schreiben aus einer Info-Variablen in eine Datei des Typs `RI` kann mit dem in dem Modul `mod_bdf_info_io` befindlichen Modulunterprogramm `BdfInfoWrite` (Moduldatei `mod_bdf_info_io.f90` im Public-Verzeichnis `record`) durch Aufruf von

```
CALL BdfInfoWrite &
     (tracefile, printfile, infofile, recfile, laktdf, &
      bdf_info, fehler)
```

in einem Schritt ausgeführt werden. Nach diesem Schritt sind alle Voraussetzungen für das Schreiben von Daten in eine BDF-Datei des Typs `DD` erfüllt.

7.7 Daten-Records Schreiben

Das Schreiben eines Daten-Records in eine BDF-Datei des Typs `DD` zerfällt in die Teilschritte Datei öffnen, Recordnummer berechnen, Daten-Record schreiben und Datei schließen. Werden mehrere Daten-Records hintereinander in dieselbe Datei geschrieben, so kann das wiederholte Öffnen und Schließen entfallen. Zum Schreiben der Daten-Records ist es erforderlich, daß zuvor die Info-Records in die korrespondierende Datei des Typs `RI` geschrieben werden (siehe Abschnitt 7.6 auf der vorherigen Seite).

Das Öffnen und Schließen einer Daten-Datei, sowie die Berechnung der Recordnummer erfolgen vollständig analog zu den in Abschnitt 7.5 auf Seite 60 gegebenen Erläuterungen.

7.7.1 Record schreiben

Bei Berechnung der Recordnummer `IREC` (siehe Abschnitt 7.5.3 auf Seite 61) wurde das Ergebnis der gleichnamigen Komponente >Recordnummer< einer BDF-Variablen `bdf_data` vom Typ `t_bdf_data` zugewiesen. Das Schreiben eines Daten-Records in eine BDF-Datei

des Typs `DD` ist mit Hilfe des in dem Modul `mod_BdfData` vorhandenen Modulunterprogrammes `write_bdf_data` (Moduldatei `mod_BdfData.f90` in dem Public-Verzeichnis `datatypes`) durch Aufruf von

```
CALL write_bdf_data &
     (tracefile, printfile, datfile, bdf_data, fehler)
```

in einfacher Weise möglich, wobei modulintern das externe Unterprogramm `dadowr.f` genutzt wird.

8 Nachwort

Das BDF-Format wird von vielen bei der BAW-AK existierenden Anwendungsprogrammen genutzt. Es folgt eine kurze Übersicht der weitestgehend dokumentierten Programme, gegliedert nach ihrem Einsatzgebiet:

- Daten-Konvertierungsprogramme:
  - ARTDIDA,
  - DIDAMINTQ (Fortran90),
  - DIDAMINTZ (Fortran90),
  - DIDATM2,
  - ENERF (Fortran90),
  - METDIDA,
  - PGCALC (Fortran90),
  - TICDIDA,
  - TM2DIDA,
  - TR2DIDA,
  - XTRDATA,
  - XTRLQ2 und
  - ZEITR.
- Hilfsprogramme:
  - DIDAMERGE,
  - DIDARENAME (Fortran90) und
  - DIDASPLIT (Fortran90).
- Graphikprogramme:
  - HVIEW2D,
  - LQ2PRO und
  - VVIEW2D.

- Simulationsprogramme:
  - PARTRACE,
  - TRIM-2D,
  - TRIM-3D und
  - WARM.
- Analyseprogramme:
  - FRQWF,
  - LZKAF (Fortran90),
  - LZKMF (Fortran90),
  - LZKSF,
  - LZKWF,
  - TDKLF,
  - TDKSF,
  - TDKVF,
  - TDKWF und
  - VTDK.

Diese Beschreibung des bei der BAW-AK benutzten BDF-Formats kann nur eine kleine zusätzliche Hilfestellung für das Verständnis der Grundlagen sowie den Umgang mit der hierzu schon vorhandenen Software sein. Zum »BDF-Experten« wird man nur durch wiederholte Nutzung in selbstgeschriebenen Anwendungsprogrammen, ganz gemäß dem Leitsatz *Learning by Doing*.

Index