DLR-IB-FT-BS-2017-33

RCDF
Institutsstandard für Flugversuchsdateien

Interner Bericht

Autor: Achim Jäkel

Deutsches Zentrum
DLR für Luft- und Raumfahrt

DLR-IB-FT-BS-2017-33

RCDF
Institutsstandard für Flugversuchsdateien

A. Jäkel

DLR
Institut für Flugsystemtechnik
Braunschweig

51 Seiten
10 Tabellen
7 Literaturstellen

Deutsches Zentrum für Luft- und Raumfahrt e.V.
Institut für Flugsystemtechnik
Abteilung Flugdynamik und Simulation

Stufe der Zugänglichkeit: I, öffentlich zugänglich

Braunschweig, März 2017

Unterschriften:

Institutsdirektor:	Prof. Dr.-Ing. S. Levedag	.....
Abteilungsleiter:	Dr.-Ing. H. Duda	.....
Verfasser:	A. Jäkel	.....

Inhaltsverzeichnis

Ziel des Dokuments

Das RCDF-Dateiformat, synonym auch als R-CDF bezeichnet, wurde 1999 im Institut für Flugsystemtechnik von Klaus Alvermann, Stephan Graeber, Achim Jäkel und Lothar Thiel als Datenschnittstelle für den ACT/FHS entwickelt und in dem Dokument ***ACT/FHS: Software Anforderungen: Schnittstellen*** [1] erstmalig beschrieben. Die aktuelle englische Version des Dokuments ***ACT/FHS: Software Requirements: Interfaces*** [2] stammt aus dem Jahr 2015. Seit der Anfangszeit hat sich der Dateistandard für weitere Messanlagen etabliert und wurde weiterentwickelt. Die in den oben referenzierten Dokumenten festgelegten Definitionen sind aber weiterhin uneingeschränkt gültig.

Dieser Bericht hier ist für alle Nutzer von Flugversuchsdaten gedacht, die mit RCDF-Dateien arbeiten. Da der Zugriff auf die ACT/FHS Dokumente nur einem eingeschränkten Nutzerkreis erlaubt ist, war es nötig, ein weiteres Dokument ohne den Kontext zu ACT/FHS zu erstellen. Neben dem grundlegenden Aufbau einer RCDF-Datei wird in diesem Bericht auch die Anwendung einiger im Institut für Flugsystemtechnik entwickelten Programme für den erleichterten Zugriff auf das RCDF-Datenformat beschrieben.

Zusammenfassung

Die Speicherung großer Flugversuchsdateien erfolgt im Institut für Flugsystemtechnik im **RCDF** (**R**aw **C**ommon **D**ata **F**ormat) Dateistandard. Dieser Standard basiert auf dem **CDF** (**C**ommon **D**ata **F**ormat) Datenformat [3], welches vom Goddard Space Flight Center der NASA entwickelt und gepflegt wird. Die CDF-Bibliothek ist frei verfügbar und kann zur Entwicklung eigener Programme verwendet werden.

Im RCDF-Standard sind bestimmte Eigenschaften der CDF-Datei festgelegt, um den Prozess der Datenauswertung zu rationalisieren. In einer RCDF-Datei werden Messdaten als ein- oder zweidimensionale Tabellen mit unterschiedlichen Datentypen und Abtastraten gespeichert. Zusätzliche Informationen werden als globale Attribute und Variablenattribute gespeichert. Der Dateizugriff ist durch die ausschließliche Verwendung der CDF-Bibliothek transparent und plattformunabhängig. Durch das binäre Speicherformat und den direkten Speicherzugriff ist die Speicherung in CDF-Dateien sehr effizient hinsichtlich Speicherbedarf und Schnelligkeit des Datenlesens.

Für den benutzerfreundlichen Zugriff auf RCDF-Dateien unter MATLAB [4] wurden im Institut für Flugsystemtechnik die Funktionen *cdf_head*, *cdf_read*, *cdf_hardread* und *cdf_write* entwickelt. Diese Schnittstellen sind in C programmiert und Teil des Programmpakets FitlabGui [5]. Außerdem stehen Konvertierungsprogramme in das RCDF-Format für Messdaten aus den Versuchsträgern ACT/FHS, ATRA, ARTIS sowie der Sensorplattform für Fallschirmversuche FDAS zur Verfügung.

1 Einleitung und Historie

Die Speicherung von Versuchsdaten in standardisierten Dateiformaten hat im Institut für Flugsystemtechnik eine lange Tradition. Der Vorteil eines fest definierten Datenstandards liegt darin, dass die Prozesse der nachfolgenden Datenauswertung vereinfacht werden können.

Im heute obsoleten Institutsstandard **IS2** wurden die Daten blockweise binär gespeichert. Mehrere Headerzeilen dienten zur Speicherung von Zuzsatzinformationen wie Signalnamen, Einheiten und Kommentaren. Der Zugriff erfolgte sequentiell. Es gab eine Begrenzung auf maximal 100 Messsignale inklusive Zeitachse sowie den Datentyp *FLOAT* (4-Byte Fließkommazahl). Eine weitere Einschränkung war die Abhängigkeit der Datendatei vom Betriebssystem und der Hardwareplattform. Mit der Verbreitung unterschiedlichster Rechnersysteme und gestiegenen Anforderungen an die Zahl der Messsignale musste ein neuer Standard etabliert werden.

Mit dem Institutsstandard **IS3** [6] wurde ab dem Jahr 1994 ein plattform- und betriebssystemunabhängiges Datenformat eingeführt. Dieser Standard benutzt das CDF-Dateiformat der NASA [3]. Datensignale werden in einer CDF-Datei als Variablen gespeichert. Es gibt Attribute für die Datei sowie für jede Variable, um Zusatzinformationen wie Einheiten und Kommentare zu speichern. Der Institutsstandard IS3 legt fest, welche Variablen und Attribute vorhanden sein müssen. Alle Messdaten in einer IS3-Datei sind vom Typ *FLOAT* (4-Byte Fließkommazahl) und auf eine gemeinsame Zeitachse interpoliert. Diese Vorverarbeitung ist jedoch der Nachteil des IS3 Standards: Durch die gemeinsame Zeitachse für alle Messsignale gehen einerseits höher abgetastete Daten verloren, während andererseits niedrig abgetastete Daten die Datei durch Interpolation unnötig vergrößern. Außerdem kann man den Daten nicht mehr ansehen, ob Quellsignale ausgefallen sind oder nicht, da immer der letzte empfangene Wert in der IS3-Datei gespeichert wird. Die Beschränkung auf den Speichertyp *FLOAT* ist für die Genauigkeit mancher Messwerte nicht ausreichend. Trotz seiner Einschränkungen ist der IS3-Standard aber noch bis heute in Benutzung.

Der für die Messanlage des ACT/FHS entwickelte und ab dem Jahr 1999 eingeführte erweiterte Institutsstandard **RCDF** [1] beseitigt die oben beschriebenen Nachteile des IS3-Standards. Er verwendet als Schnittstelle ebenfalls die CDF-Bibliothek. Versuchsdaten werden in ein- oder zweidimensionalen Tabellen gespeichert. Die Daten werden dabei weder interpoliert noch gekürzt. Verschiedene Datentypen sind möglich, um keine Information in den Daten zu verfälschen. Die Messsignale im RCDF-Standard haben keine äquidistante Zeitachse, da jeder Messwert zusammen mit dem Zeitpunkt seiner Erfassung abgespeichert wird. Dadurch sind Lücken in der Aufzeichnung von Quellsignalen weiterhin erkennbar. Eine Interpolation der Datensignale auf eine gemeinsame Zeitachse und eine Umwandlung in den Typ *DOUBLE* (8-Byte Fließkommazahl) kann beim Lesen der Daten durch geeignete Programme (Abschnitte 4.3, 4.4, 4.5) erfolgen. In diesem Fall werden die Daten jedoch teilweise verfälscht, so dass die eigentlichen Vorteile des RCDF-Formats verloren gehen.

2 Die CDF-Bibliothek

CDF [3] ist eine abstrakte Schnittstelle zur Speicherung und Manipulation von Versuchsdaten, welche vom Goddard Space Flight Center der NASA entwickelt und gepflegt wird. Die CDF-Bibliothek sowie verschiedene Werkzeuge sind frei verfügbar. Unter dem folgenden Link sind alle Informationen zur CDF-Bibliothek zu finden:

https://cdf.gsfc.nasa.gov/

Der Nutzer von CDF benötigt keine Fachkenntnisse über die physikalische Speicherung der Daten und das Dateiformat, welches dahinter liegt, da der Datenzugriff ausschließlich über die CDF-Bibliothek erfolgt. CDF-Dateien sind unabhängig vom Betriebssystem und können ohne Umwandlung zwischen verschiedenen Systemen ausgetauscht werden.

In einer CDF-Datei werden Versuchsdaten als Variablen in ein- oder mehrdimensionalen Feldern abgespeichert. Verschiedene Datentypen (Integer, Fließkommazahl, Character, …) sind möglich. Globale Attribute und Variablenattribute erlauben die Speicherung von Zusatzinformationen.

Die CDF-Bibliothek ist auf Geschwindigkeit und Speicherplatzersparnis optimiert. Sie bietet eine programmierfreundliche Möglichkeit, große Datenmengen effizient zu verwalten. Diese Vorteile haben dazu geführt, dass das CDF-Dateiformat seit mehr als 20 Jahren Grundlage für die Versuchsdatenanalyse im Institut für Flugsystemtechnik ist.

3 Der Institutsstandard RCDF

Das RCDF-Dateiformat, synonym auch als R-CDF bezeichnet, wird in dem Dokument ***ACT/FHS: Software Requirements: Interfaces*** [2] aus dem Jahr 2015 definiert. Dieser Abschnitt hier fasst die grundlegenden Eigenschaften einer RCDF-Datei zusammen und weist auf Erweiterungen hin. Detailliertere Beschreibungen sind dem Originaldokument zu entnehmen.

Im RCDF-Standard ist festgelegt, wie Variablen und Attribute in einer CDF-Datei zu speichern sind. Es gibt Attribute mit globaler Gültigkeit (*GLOBAL_SCOPE*) und lokaler Gültigkeit für einzelne Variablen (*VARIABLE_SCOPE*). Messsignale werden in Variablen (*zVAR*) gespeichert. Sie werden über ihren eindeutigen Namen identifiziert, eine feste Reihenfolge gibt es nicht.

Die richtige Wahl der Dateiattribute beim Erstellen einer CDF-Datei gewährleistet eine effektive Datenspeicherung. Insbesondere können abweichende Einstellungen dazu führen, dass sich die Lesegeschwindigkeit verschlechtert. Die Einstellung für *CDF_FORMAT* muss für RCDF-Dateien immer *SINGLE_FILE* sein.

Option	Wert	Bedeutung
CDF_ENCODING	HOST_ENCODING	Kodierung für lokales Betriebssystem
CDF_MAJORITY	COL_MAJOR	Variablen spaltenweise speichern
CDF_FORMAT	SINGLE_FILE	Alle Variablen in einer Datei speichern

Tabelle 1: Dateiattribute

3.1 Globale Attribute

Globale Attribute enthalten Informationen, die für die gesamte Datei (*GLOBAL_SCOPE*) gültig sind. Dazu gehören Informationen zum Dateistandard, zur Zeitbasis und Textinformationen zur Herkunft der Datei. Nur die **fett markierten Attribute** werden durch die weiter unten beschriebenen Leseprogramme (Abschnitte 4.3, 4.4, 4.5) ausgewertet.

Attribut	Typ	Bedeutung
TITLE	CDF_CHAR	Dateititel
TYPE	CDF_CHAR	Dateistandard („RCDF“)
VERSION	CDF_CHAR	Versionsinformation
UTCTIME	CDF_EPOCH	Startzeit der Messdaten
DMCCYCLE	CDF_REAL4	Zyklusdauer in Sekunden
BEGCYCLE	CDF_INT4	Zykluszähler zum Zeitpunkt UTCTIME
COMMENT	CDF_CHAR	Freier Kommentar
FILENAME	CDF_CHAR	Name der Datei ohne Pfad und Typ
HDR_FILENAME	CDF_CHAR	Name der Headerdatei mit weiteren Informationen (ACT/FHS spezifisch)
PARENT_FILENAME	CDF_CHAR	Name einer verknüpften RCDF-Datei (**optional**)

Tabelle 2: Globale Attribute

Das Attribut *TYPE* dient zum Erkennen des richtigen Dateistandards und muss im RCDF-Standard den Text „RCDF“ enthalten.

Die Attribute *UTCTIME*, *DMCCYCLE* und *BEGCYCLE* sind erforderlich zur Berechnung einer Zeitachse für die Messsignale. Eine RCDF-Datei hat keine einheitliche und äquidistante Zeitachse für alle Signale, um Daten nicht schon beim Speichern zu verfälschen. Stattdessen wird zu jedem Messwert ein Zykluszähler gespeichert, mit dem beim Lesen aus der Datei ein Zeitpunkt in Sekunden berechnet werden kann. Das genaue Verfahren zur Berechnung der Zeitachse wird weiter unten (Abschnitt 3.5) beschrieben.

Die Attribute *FILENAME* und *HDR_FILENAME* gab es im ursprünglichen Entwurf des RCDF-Standards [1] noch nicht, sie kamen im Jahr 2001 hinzu.

Das optionale Attribut *PARENT_FILENAME* ist ebenfalls eine Erweiterung des RCDF-Dateiformats [2]. Ist dieses Attribut vorhanden, wird dem Leseprogramm damit mitgeteilt, dass es noch weitere Datensignale in einer angehängten RCDF-Datei gibt. Der Dateiname dieser Datei muss ohne Dateipfad und Dateityp angegeben werden, da vorausgesetzt wird, dass sich die beiden verknüpften Dateien im selben Verzeichnis befinden. Mit diesem Attribut hat der Nutzer die Möglichkeit, eigene Datensignale an eine Datei mit Nur-Lese-Rechten anzuhängen, ohne die Originaldatei zu verändern.

3.2 Speicherung von Messdaten in Variablen

In einer RCDF-Datei werden Messdaten als Variablen (*zVAR*) gespeichert, entweder als Basisdatentyp oder als eindimensionales Feld eines Basisdatentyps. Jede Variable hat einen eindeutigen Namen. Es gibt fest vorgegebene Variablen für Zykluskanäle und Bitsignale sowie frei definierbare Variablen für die eigentlichen Messdaten. Die folgenden Eigenschaften einer Variablen werden bei deren Erstellung festgelegt:

Eigenschaft	Wert	Bedeutung
zVAR_NAME	*VarName*	Eindeutiger Name der Variablen (maximal 63 Zeichen)
zVAR_DATATYPE	CDF-Basisdatentyp	Datentyp der Variablen
zVAR_NUMELEMS	1	Anzahl der Elemente für jeden Datenwert (nur für Strings relevant, sonst immer 1)
zVAR_NUMDIMS	0 / 1	Anzahl der Dimensionen: 0 = Signal ist ein Basisdatentyp 1 = Signal ist ein eindimensionales Feld
zVAR_DIMSIZES	[] / [*n*]	Größe der Dimensionen: [] = Signal ist ein Basisdatentyp [*n*] = Signal ist ein eindimensionales Feld mit *n* Elementen
zVAR_RECVARY	VARY	Record Varianz der Variablen
zVAR_DIMVARYS	[] / [VARY]	Dimension Varianz der Variablen [] = Signal ist ein Basisdatentyp [VARY] = Signal ist ein eindimensionales Feld

Tabelle 3: Variableneigenschaften

3.3 Variablenattribute

Mit Variablenattributen werden Zusatzinformationen für jede Variable festgelegt:

Attribut	Typ	Bedeutung
SIGNALID	CDF_INT4	Eindeutige Nummer (Bezeichner) des Messsignals
SIGUNIT	CDF_CHAR	Einheit des Messsignals
CYCLECHN	CDF_INT4	Nummer *n* des Zykluskanals _CYCLE*n*

Tabelle 4: Variablenattribute

Das Attribut *CYCLECHN* enthält die Nummer des Zykluskanals, der dieser Variablen zugeordnet ist. Aus den Zykluswerten kann mit Hilfe der globalen Attribute *UTCTIME*, *DMCCYCLE* und *BEGCYCLE* die Zeitachse für das Messsignal generiert werden (Abschnitt 3.5).

3.4 Spezielle Variablen mit festen Namen

RCDF-Dateien enthalten einige spezielle Variablen mit festen Namen. Zu ihnen gehören die Zykluskanäle sowie Variablen zur Beschreibung von Bitsignalen.

Variable	Typ	Bedeutung
_CYCLE*n*	CDF_INT4	Zykluskanal Nummer *n*
BIT_SIGNAL_ID	CDF_INT4	Eindeutige Nummer (Bezeichner) des Bitsignals
BIT_SIGNAL_NAME	CDF_CHAR	Name des Bitsignals (64 Zeichen, mit Blanks aufgefüllt und nicht Null-terminiert)
BIT_SIGNAL_SRCID	CDF_INT4	Nummer des Quellsignals (Attribut SIGNALID des Messsignals)
BIT_SIGNAL_MASK	CDF_UINT4	Bitmaske zum Herausschneiden aus dem Quellsignal (Bits mit 1 besetzt werden herausgeschnitten, Bits mit 0 werden ignoriert)

Tabelle 5: Spezielle Variablen

Zykluskanäle sind vom Typ *CDF_INT4* und enthalten zu jedem Datenpunkt der zugeordneten Messsignale einen ganzzahligen Zykluszähler. Mit Hilfe des Zählers kann von einem Leseprogramm ein Zeitkanal *Time* in Sekunden generiert werden. Der Zykluszähler beginnt mit dem in *BEGCYCLE* gespeicherten Wert zum Zeitpunkt *UTCTIME* und wird alle *DMCCYCLE* Sekunden um 1 erhöht. Das Verfahren zur Berechnung einer Zeitachse wird weiter unten (Abschnitt 3.5) beschrieben. Gruppen von Signalen können dem gleichen Zykluskanal zugeordnet werden, wenn sie die gleiche Zeitbasis haben, was bei Messdaten aus derselben Sensornachricht der Fall ist. Für Zykluskanäle sind keine Einträge in den Variablenattributen erforderlich.

Bitsignale sind Signale, die einzelne Bits aus einem Quellsignal herausschneiden. Aus Speicherplatzgründen wird nicht jedes Bitsignal als einzelne Variable gespeichert, sondern mehrere Bitsignale werden in einem Quellsignal zusammengefasst. Für jedes Bitsignal gibt es jeweils einen Eintrag (*zENTRY)* in den Variablen *BIT_SIGNAL_ID*, *BIT_SIGNAL_NAME*, *BIT_SIGNAL_SRCID* und *BIT_SIGNAL_MASK*. Wenn ein Bitsignal gelesen wird, wird als erstes die Beschreibung gelesen, dann das Quellsignal gelesen und daraus das Bitsignal durch Ausschneiden der entsprechenden Bits und Nach-Rechts-Schieben berechnet.

3.5 Verfahren zur Berechnung der Zeitachse

In einer RCDF-Datei gibt es keine gemeinsame äquidistante Zeitachse für alle Signale, sondern es wird zu jedem Messwert der Zeitpunkt der Messung gespeichert. Die Berechnung einer gemeinsamen Zeitachse *Time* in Sekunden für mehrere Messsignale, sofern sie denn benötigt wird, ist Aufgabe eines Leseprogramms (Abschnitt 4.4).

Jedem Messsignal ist ein Zykluskanal zugeordnet, wobei mehrere Messsignale mit gleicher Zeitbasis einen gemeinsamen Zykluskanal haben können. In einem Zykluskanal ist zu jedem Datenpunkt der zugeordneten Messsignale ein Zykluswert gespeichert. Der Zykluswert ist ein ganzzahliger Zähler und wird während einer Messreihe mit dem Takt *DMCCYCLE* hochgezählt. Beim Eintreffen eines Messwerts in der Datenaufzeichnung wird der aktuelle Zykluswert zu dem Messwert in den Zykluskanal dieses Messwerts geschrieben. Es ist gültig, wenn zu mehreren aufeinanderfolgenden Signalwerten derselbe Zykluszähler gespeichert wird. Dies bedeutet, dass die Signalwerte im selben Zyklus erfasst wurden. Die Zeiten aufeinanderfolgender Signalwerte müssen also nicht streng monoton wachsend sein, sondern nur monoton wachsend.

Der Vorteil dieses Verfahrens ist, dass Messwerte mit einer Zeitgenauigkeit von beispielsweise 1000 Hz aufgelöst werden können, ohne dass tatsächlich 1000 Werte/Sekunde in der RCDF-Datei gespeichert werden müssen. Dies ergibt eine Speicherplatzersparnis gegenüber einer fixen Zeitachse für alle Signale.

Die Zeit in Sekunden zu einem Messwert wird folgendermaßen berechnet:

Time[i] = UTCTIME + (_CYCLEn[i] – BEGCYCLE) * DMCCYCLE

i	: Datensatznummer (*zENTRY*) des Messwerts
Time[i]	: Zeitpunkt des Messwerts *i* in Sekunden
UTCTIME	: Startzeit der Messreihe
_CYCLEn[i]	: Zykluswert zum Messwert *i* im Zykluskanal *n*
BEGCYCLE	: Zykluswert zum Zeitpunkt *UTCTIME* (Beginn der Messreihe)
DMCCYCLE	: Länge eines Zyklus (Taktrate) in der Datenaufzeichnung in Sekunden

4 Programme für RCDF-Dateien

Das Schreiben und Lesen von RCDF-Dateien ist nicht trivial. Für das Lesen und Interpretieren der Daten aus einer RCDF-Datei sind zahlreiche Aufrufe der CDF-Bibliotheksfunktionen nötig, um das gewünschte Ergebnis zu erhalten. MATLAB [4] bietet ebenfalls eine Schnittstelle zu den CDF-Basisroutinen an. Ein sehr nützliches Werkzeug, welches auf der CDF-Homepage zu finden ist, ist ***CDFfsi*** (CDF Full Screen Interface). Mit diesem Programm können einzelne Attribute und Daten einer CDF-Datei angesehen und geändert werden.

Die allgemeine CDF-Bibliothek bietet nur eine Basisfunktionalität für den Zugriff auf CDF-Dateien, berücksichtigt jedoch nicht die Besonderheiten des RCDF-Dateistandards. Um den Nutzer der Daten von Programmieraufwand zu entlasten, wurden im Institut für Flugsystemtechnik Programme entwickelt, mit denen man RCDF-Dateien erzeugen und von MATLAB aus komfortabel auf die Daten in einer RCDF-Datei zugreifen kann. Nachfolgend werden einiger dieser Programme vorgestellt.

4.1 Programme zur Datenkonvertierung

In den Messanlagen werden unterschiedliche Rohdatenformate abgespeichert. Aus diesem Grund gibt es auch spezifische Konvertierungsprogramme für jede Messanlage, um aus den aufgezeichneten Rohdaten RCDF-Dateien zu erzeugen und sie den Nutzern für eine anschließende Datenauswertung standardisiert zur Verfügung zu stellen.

Versuchsanlage	Konvertierungsprogramm
ACT/FHS	RexData [7]
ATRA	ATRA_konvert
FDAS	FDAS_konvert
ARTIS	artis2rcdf

Tabelle 6: Konvertierungsprogramme

Die Pflege und Anwendung dieser Konvertierungsprogramme obliegt den Verantwortlichen für die Datenaufbereitung der jeweiligen Versuchsanlage. Sie sorgen für die Datenkonvertierung nach RCDF und stellen die Dateien den Nutzern zur Verfügung.

4.2 FitlabGui

FitlabGui [5] ist ein Programmpaket für die Datenauswertung in der Entwicklungsumgebung MATLAB [4]. Es bietet zahlreiche Werkzeuge und Plotprogramme. Das Dateiformat RCDF wird in FitlabGui automatisch erkannt, gelesen und kann auch geschrieben werden. Dafür enthält FitlabGui die komfortablen Servicefunktionen *cdf_head* (Abschnitt 4.3), *cdf_read* (Abschnitt 4.4), *cdf_hardread* (Abschnitt 4.5) und *cdf_write* (Abschnitt 4.6), welche als MATLAB-mex-Functions in C programmiert sind. Diese Funktionen können vom Nutzer wie normale MATLAB-Befehle aufgerufen auch direkt in eigenen MATLAB-Scripts verwendet werden. FitlabGui steht allen Mitarbeitern des Instituts zur Verfügung.

4.3 cdf_head

Die Funktion *cdf_head* ist Bestandteil des Programmpakets FitlabGui. Sie liest Dateiinformationen wie Signalnamen, Einheiten und die Auflösung der Zeitstempel einer CDF-Datei, die dem Standard *RCDF*, *IS3* oder *AIRBUS* entspricht. Die Aufrufsyntax in MATLAB lautet:

[*Type*, *RecordRate*, *Runs*, *VarNames*, *VarUnits*] = cdf_head(*FileName*)

Parameter	Typ	Bedeutung
FileName	CHAR[*]	Name der CDF-Datei
Type	CHAR[8]	Dateityp (*RCDF*, *IS3*, *AIRBUS* oder *UNKNOWN*)
RecordRate	DOUBLE	Abtastrate (*DMCCYCLE* bei RCDF-Dateien)
Runs	DOUBLE[*n*,2]	Matrix mit Anfangs- und Endzeiten der *n* Versuchsreihen (nur IS3-Dateien können mehrere Runs enthalten)
VarNames	CHAR[*n*,*]	Matrix mit *n* Namen der Signale (Signalliste)
VarUnits	CHAR[*n*,*]	Matrix mit *n* Einheiten der Signale

Tabelle 7: Parameter cdf_head

Die Funktion ist auf maximal 10000 Signalnamen begrenzt. Da RCDF-Dateien keinen expliziten Zeitkanal enthalten, liefert die Funktion als erste Zeile in der Signalliste (*VarNames*) einen virtuellen Zeitkanal mit dem Namen „*Time*" und als erste Zeile in *VarUnits* die Einheit „*s*" zurück. Danach folgen die Namen aller Bitsignale als einzelne Signalnamen und schließlich die Namen aller übrigen Messsignale. Die Namen der Zykluskanäle sowie die Namen der vier Variablen für die Beschreibung der Bitsignale (Abschnitt 3.4) sind nicht Bestandteil der zurückgegebenen Signalliste. Handelt es sich bei einem Messsignal um eine zweidimensionale Tabelle, so wird für jedes Element der zweiten Dimension ein Name in der Form *VarName*(*i*) für das *i*-te Element von *VarName* erzeugt. Zu jeder Zeile in der Signalliste *VarNames* gibt es eine Zeile in der Einheitenliste *VarUnits* mit der entsprechenden Einheit des Signals. Falls in einer RCDF-Datei das globale Attribut *PARENT_FILENAME* existiert, enthält die Signalliste erst alle Signale aus der verknüpften Elterndatei und danach die Signale aus der Kinddatei.

4.4 cdf_read

Die Funktion *cdf_read* ist Bestandteil des Programmpakets FitlabGui. Mit ihr können mehrere Messsignale gleichzeitig aus einer CDF-Datei gelesen werden, die dem Standard *RCDF*, *IS3* oder *AIRBUS* entspricht. Die Aufrufsyntax in MATLAB lautet:

[*Data*,*Rdt*] = cdf_read(***FileName***,*VarNames*,*TS*,*TE*,*RD*)

Parameter	Typ	Bedeutung
FileName	CHAR[*]	Name der CDF-Datei
VarNames	CHAR[*n*,*]	Matrix mit *n* Namen der zu lesenden Signale (falls nicht angegeben oder [], werden alle Signale gelesen)
TS, TE	DOUBLE	Zeitbereich, der gelesen werden soll (falls nicht angegeben oder -1, -2, wird der gesamte Zeitbereich gelesen)
RD	INTEGER	Datenreduktionsfaktor (nur jeder *RD*-te Wert wird gelesen. Default=1)
Data	DOUBLE[*n*,*m*]	Signaldaten (*n* Datenpunkte für *m* Signale)
Rdt	DOUBLE	Neue Abtastrate nach Datenreduzierung

Tabelle 8: Parameter cdf_read

Die Funktion kann maximal 5000 Signale gleichzeitig lesen. Falls in einer RCDF-Datei das globale Attribut *PARENT_FILENAME* existiert, wird zuerst die verknüpfte Elterndatei nach den angeforderten Signalen durchsucht und danach die Kinddatei.

Der Aufruf der Funktion *cdf_read* benötigt als einzigen zwingenden Eingabeparameter *FileName*, die anderen Parameter können weggelassen werden. Wird einer der hinteren Parameter benötigt, kann die Funktion mit Platzhaltern für die vorderen Parameter aufgerufen werden. Platzhalter sind *VarNames*=[], *TS*=-1 und *TE*=-2. *TS* und *TE* sind nur zusammen als Paar gültig.

Ein Aufruf von *cdf_read* mit dem einzigen Parameter *FileName* bewirkt, dass die Daten aller Signale über die gesamte Messzeit mit der kleinsten Zeitauflösung (bei RCDF-Dateien *DMCCYCLE*) gelesen werden. Da RCDF-Dateien sehr groß und die Taktrate sehr klein sein kann, kann die zurück gelieferte Datenmenge enorm groß und die Lesezeit lang werden. Besser ist es, nur die tatsächlich benötigten Signale und den benötigten Zeitbereich anzufordern, wozu die Parameter *VarNames*, *TS* und *TE* dienen. In einer RCDF-Datei gibt es keine Zeitachse. Sollte diese trotzdem benötigt werden, so muss sie mit dem virtuellen Signalnamen „*Time*" angefordert werden und wird dann von *cdf_read* berechnet (Abschnitt 3.5).

Mit dem Datenreduktionsfaktor *RD* wird das Abtastintervall vergrößert und damit die zurückgelieferte Datenmenge ebenfalls reduziert. So wird zum Beispiel mit einem *RD*=20 und *DMCCYCLE*=0.001Sekunden das Abtastintervall der zurückgelieferten Daten auf *Rdt*=0.02 Sekunden vergrößert. Der Nutzer des Leseprogramms *cdf_read* gibt den Zeitbereich und den gewünschten Reduktionsfaktor vor, und die Funktion interpoliert

die vorhandenen Daten auf diese Zeitachse. Dabei werden Fließkommazahlen linear interpoliert, während ganzzahlige Daten mit der Methode Sample-And-Hold interpoliert werden. Ist eine Extrapolation nötig, wird nach vorne mit dem ersten Datenpunkt und nach hinten mit dem letzten Datenpunkt extrapoliert. Angeforderte Bitsignale werden anhand der Bitmaske aus dem Quellsignal ausgeschnitten.

Zurückgeliefert wird eine Datenmatrix vom Typ *DOUBLE[n,m]* (8-Byte Fließkommazahlen) mit *n* Datenpunkten für *m* Signale. Dabei ist zu beachten, dass in *cdf_read* die Ursprungsdaten durch Interpolation, Weglassen und Typkonvertierung verfälscht werden können. Ein temporärer Signalausfall in der Messreihe ist danach nicht mehr erkennbar.

4.5 cdf_hardread

Die Funktion *cdf_hardread* ist Bestandteil des Programmpakets FitlabGui. Sie dient zum Lesen eines einzelnen Messsignals aus einer RCDF-Datei. Im Unterschied zu *cdf_read* (Abschnitt 4.4) werden die Daten hierbei jedoch nicht durch Interpolation oder Weglassen verfälscht, sondern so ausgegeben, wie sie bei der Messung aufgezeichnet wurden. Dadurch bleiben sowohl der originale Zeitverlauf als auch eventuell vorhandene Lücken in der Messreihe erhalten. Die Aufrufsyntax in MATLAB lautet:

[*Data*,*IntData*] = cdf_hardread(***FileName***,***VarName***,*TS*,*TE*)

Parameter	Typ	Bedeutung
FileName	CHAR[*]	Name der RCDF-Datei
VarName	CHAR[*]	Name des zu lesenden Signals
TS, TE	DOUBLE	Zeitbereich, der gelesen werden soll (falls nicht angegeben oder -1, -2, wird der gesamte Zeitbereich gelesen)
Data	DOUBLE[*n*,2]	Zeit- und Signaldaten (*n* Datenpunkte)
IntData	DOUBLE	Schalter: 1=Messsignal ist ein ganzahliger Datentyp 0=Messsignal ist ein Fließkomma Datentyp

Tabelle 9: Parameter cdf_hardread

Die Parameter *FileName* und *VarName* sind zwingend erforderlich. Falls die optionalen Parameter *TS* und *TE* nicht angegeben oder mit den Platzhaltern *TS*=-1 und *TE*=-2 besetzt sind, wird das gesamte Messintervall gelesen. TS und TE sind immer nur zusammen als Paar gültig. Existiert in der RCDF-Datei das globale Attribut *PARENT_FILENAME*, so wird als erstes die Elterndatei nach dem angeforderten Messsignal durchsucht und danach die Kinddatei.

Die Funktion gibt eine zweispaltige Matrix *Data* mit dem Datentyp *DOUBLE* (8-Byte Fließkommazahl) zurück, wobei die erste Spalte die Zeitachse in Sekunden und die zweite Spalte die Messdaten des ausgewählten Signals enthält. Der Rückgabeparameter *IntData* ist auf 1 gesetzt, falls das Messsignal ein ganzzahliger Datentyp ist und auf 0 für Fließkommazahlen.

4.6 cdf_write

Die Funktion *cdf_write* ist Bestandteil des Programmpakets FitlabGui. Mit ihr können CDF-Dateien geschrieben werden, die dem Standard *RCDF* oder *IS3* entsprechen. Die Aufrufsyntax in MATLAB lautet:

cdf_write(***FileName***,***Type***,***VarNames***,***Data***,*Units*)

Parameter	Typ	Bedeutung
FileName	CHAR[*]	Name der CDF-Datei, die erzeugt werden soll
Type	CHAR[4]	Dateityp (*RCDF* oder *IS3*)
VarNames	CHAR[*n*,*]	*n* Namen der zu schreibenden Signale
Data	DOUBLE[*m*,*n*]	Daten für *n* Signale mit jeweils *m* Datenpunkten
Units	CHAR[*n*,*]	Einheiten für *n* Signale

Tabelle 10: Parameter cdf_write

Eine CDF-Datei mit dem Namen aus *FileName* wird erzeugt. Falls eine Datei mit demselben Namen bereits existiert, wird diese überschrieben. In der Datenmatrix *Data* muss die erste Spalte immer die Zeitachse sein, ebenso wie die erste Zeile in *VarNames*. Für den Dateityp *IS3* ist es erforderlich, dass die Zeitachse eine konstante Abtastrate hat. Größere Sprünge in der Zeitachse werden in IS3-Dateien als separate Zeitabschnitte interpretiert. Bei RCDF-Dateien gibt es diese Einschränkung nicht. Aber durch die gemeinsame Zeitachse für alle Signale werden die Vorteile einer RCDF-Datei an dieser Stelle nicht ausgenutzt, da eine sehr spezielle RCDF-Datei erzeugt wird.

Der Parameter *Units* ist optional und kann weggelassen werden. Die Funktion kann maximal 10000 Signale inklusive Zeitachse schreiben.

5 Ausblick

Der Institutsstandard RCDF hat sich seit seiner Einführung vor 18 Jahren praktisch bewährt. Diese lange Zeitspanne zeigt, wie weitsichtig der Entwurf schon damals war. Er erfüllt bis heute alle Anforderungen an eine effiziente Datenspeicherung.

Die Erzeugung von RCDF-Dateien erfolgt mit speziellen Konvertierungsprogrammen, die an die jeweilige Messanlage und deren Rohformat angepasst sind. Sie sind Teil des Prozesses für die Messdatenaufbereitung der Forschungsplattformen ACT/FHS, ATRA, ARTIS und der Sensorplattform für Fallschirmversuche FDAS. Mit der Einführung einer neuen Messanlage entsteht auch Entwicklungsbedarf für ein neues Konvertierungsprogramm für RCDF-Dateien, um die Daten für die Nutzer aufzubereiten.

Zurzeit erfolgt die Datenauswertung im Institut für Flugsystemtechnik mit FitlabGui unter MATLAB. Hier gibt es komfortable Funktionen zum Zugriff auf RCDF-Dateien, wie oben beschrieben. MATLAB selber bietet ebenfalls Basisfunktionen, um CDF-Dateien zu lesen. Diese sind jedoch in der Handhabung komplizierter und deswegen zum Lesen von RCDF-Dateien nicht zu empfehlen.

Außerhalb von MATLAB gibt es generische Funktionen in C, mit denen man auf CDF-Dateien zugreifen kann sowie verschiedene Werkzeuge, die man von der CDF-Homepage (siehe Kapitel 2) herunterladen kann. Mit Funktionen aus der CDF-Bibliothek können sich Nutzer eigene Programme in C/C++ erstellen, um auf RCDF-Dateien zuzugreifen.

Literaturverzeichnis

[1] K. Alvermann, M. Bodenstein, S. Graeber, H. Oertel und L. Thiel, „ACT/FHS: Software Anforderungen: Schnittstellen. SWE L 220R0416 D01," DLR, Braunschweig, 1999.

[2] K. Alvermann, M. Bodenstein, S. Graeber, H. Oertel und L. Thiel, „ACT/FHS: Software Requirements: Interfaces. SWE L 220R0416 D01," DLR, Braunschweig, 2015.

[3] Space Physics Data Facility NASA/Goddard Space Flight Center, „CDF User's Guide Version 3.6.2," Greenbelt, Maryland 20771 (U.S.A.), 2016.

[4] N.N., „MATLAB Documentation R2012b," The MathWorks, Inc., 1994-2012.

[5] S. Seher-Weiß, „FitlabGui A MATLAB Tool for Flight Data Analysis and Parameter Estimation Version 2.5. IB 111-2015/34," DLR, Braunschweig, 2015.

[6] J. Buchholz und G. Wulff, „Institutsstandard IS3. IB 111-94/35," DLR, Braunschweig, 1994.

[7] K. Alvermann, M. Bodenstein, H.-U. Döhler, S. Graeber und H. Oertel, „Software User Documentation. SWE L 220R0420 D01," DLR, Braunschweig, 2016.

Anhang

A.1 cdf_head.c

```
/*
    cdf_head.c  .MEX file implementation for reading
                the standard header information of an
                CDF file (IS3, FRG, RCDF, AIRBUS).

    Author: Achim.Jaekel@dlr.de
    Date  : 24.September 2002

    The calling syntax is:
    [filetype,record_rate,runs,varnames,units] = cdf_head(filename)

    cdf_head opens the CDF file specified by filename and returns
    the header information required to retrieve data.

    INPUT:
    filename     - string containing the name of the RCDF file.

    OUTPUT:
    filetype     - file type (IS3, FRG, RCDF, AIRBUS)
    record_rate  - scanning rate resp. DMC cycle time
    runs         - start/end times of runs
    varnames     - names of variables
    units        - dimensions
*/

/* Includes*/
#include <math.h>
#include <string.h>
#include "mex.h"
#include "cdf_sublib.h"

/* Constants*/
#define MAXVAR 10000
#define UNIT_LEN 128

/* Arguments*/
#define FILENAME    prhs[0]
#define FILETYPE    plhs[0]
#define RECORD_RATE plhs[1]
#define RUNS        plhs[2]
#define VARNAMES    plhs[3]
#define UNITS       plhs[4]

void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{
    char      filename[CDF_PATHNAME_LEN+1]; /* Pathname of RCDF file */
    char      filetype[8] = {"UNKNOWN"};    /* Type of cdf file */
    double    *record_rate;                 /* Scanning rate */
    double    *runs;                        /* Runs start/end times */
    char      *varnames[MAXVAR];            /* Variable names */
    char      *units[MAXVAR];               /* Dimensions */
    float     t[2];
    double    *time,UTCTime,tOffset;
    int       i,k,iFile,BegCycle,MinCycleNr,MaxCycleNr;
    long      nVars,Len,n,year,month,day,hour,minute,second;
    long      NumDims,DimSizes[2],indices[2],counts[2],intervals[2];
    CDFstatus status;
    CDFid     Id[2];
    char      *pt,TextBuffer[1024],NumStr[8];
    char      VarNameBuffer[MAXVAR][CDF_VAR_NAME_LEN+8];
    char      UnitBuffer[MAXVAR][UNIT_LEN+1];

    /* Check argument */
    if (nrhs < 1 || (!mxIsChar(FILENAME))) {
```

```
        mexErrMsgTxt("cdf_head: Filename argument required");
    }
    if (mxGetN(FILENAME) > CDF_PATHNAME_LEN) {
        mexErrMsgTxt("cdf_head: Filename too long");
    }
    mxGetString(FILENAME,filename,CDF_PATHNAME_LEN+1);

    /* Open file */
    status = CDFlib(  OPEN_,CDF_,filename,&Id[0],
                    SELECT_,CDF_READONLY_MODE_,READONLYon,NULL_);
    if (status != CDF_OK) disperr(status);

    /* Get TYPE of the file (DLR-FT standards)*/
    status = CDFlib(SELECT_,ATTR_NAME_,"TYPE",
                          gENTRY_,0,
                      GET_,gENTRY_DATA_,TextBuffer,NULL_);
    if (status == CDF_OK) {
        if (strncmp(TextBuffer,"IS3",3) == 0) strcpy(filetype,"IS3");
        if (strncmp(TextBuffer,"FRG",3) == 0) strcpy(filetype,"FRG");
        if (strncmp(TextBuffer,"RCDF",4) == 0) strcpy(filetype,"RCDF");

    /* AIRBUS standard?*/
    } else {
        status = CDFlib(CONFIRM_,ATTR_EXISTENCE_,"SAMPLING_PERIOD(S)",NULL_);
        if (status == CDF_OK) strcpy(filetype,"AIRBUS");
    }

    /* Get number of zVariables */
    CDFlib(GET_,CDF_NUMzVARS_,&nVars,NULL_);
    if (nVars > MAXVAR) {
        CDFlib(CLOSE_,CDF_,NULL_);
        mexErrMsgTxt("cdf_head: Number of variables exceeds program limit");
    }

    /* Interface*/
    FILETYPE    = mxCreateString((const char*)filetype);
    RECORD_RATE = mxCreateDoubleMatrix(1,1,mxREAL);
    record_rate = mxGetPr(RECORD_RATE);

/*****************************************/
/* Get header information from IS3/FRG file */
/*****************************************/
    if ((strcmp(filetype,"IS3") == 0) || (strcmp(filetype,"FRG") == 0)) {

        /* Get RECORD_RATE */
        CDFlib(SELECT_,ATTR_NAME_,"RECORD_RATE",
                       gENTRY_,0,
                  GET_,gENTRY_DATA_,t,NULL_);
        *record_rate = t[0];

        /* Get RUNS */
        CDFlib(SELECT_,ATTR_NAME_,"RUNS",
                  GET_,ATTR_NUMgENTRIES_,&n,NULL_);
        RUNS = mxCreateDoubleMatrix(n,2,mxREAL);
        runs = mxGetPr(RUNS);
        for (i=0 ; i<n ; i++) {
            CDFlib(SELECT_,gENTRY_,i,
                       GET_,gENTRY_DATA_,t,NULL_);
            *(runs+i)   = t[0];
            *(runs+n+i) = t[1];
        }

        /* Get Variable Names and Units */
        CDFlib(SELECT_,ATTR_NAME_,"UNITS",NULL_);
        for (i=0 ; i<nVars ; i++) {
            CDFlib(SELECT_,zVAR_,i,
                       GET_,zVAR_NAME_,VarNameBuffer[i],NULL_);
            varnames[i] = deblank(VarNameBuffer[i]);
            status = CDFlib(SELECT_,zENTRY_,i,
                                GET_,zENTRY_DATA_,UnitBuffer[i],
                                     zENTRY_NUMELEMS_,&Len,NULL_);
            if (status != CDF_OK) Len = 0;
            if (Len > UNIT_LEN) Len = UNIT_LEN;
            UnitBuffer[i][Len] = '\0';
```

```
            units[i] = deblank(UnitBuffer[i]);
        }

/*************************************/
/* Get header information from RCDF file */
/*************************************/
    } else if (strcmp(filetype,"RCDF") == 0) {

        /* Get PARENT_FILENAME */
        status = CDFlib(CONFIRM_,ATTR_EXISTENCE_,"PARENT_FILENAME",NULL_);
        if (status == CDF_OK) {
            pt = strrchr(filename,filesep());
            if (pt == NULL) {
                pt = filename;
            } else {
                pt++;
            }
            CDFlib(SELECT_,ATTR_NAME_,"PARENT_FILENAME",
                           gENTRY_,0,
                     GET_,gENTRY_DATA_,pt,
                           gENTRY_NUMELEMS_,&Len,NULL_);
            *(pt+Len) = '\0';

            /* Open parent file */
            status = CDFlib(  OPEN_,CDF_,filename,&Id[1],
                            SELECT_,CDF_READONLY_MODE_,READONLYon,NULL_);
            if (status != CDF_OK) {
                CDFlib(SELECT_,CDF_,Id[0],
                        CLOSE_,CDF_,NULL_);
                printf("PARENT_FILENAME = %s\n",filename);
                disperr(status);
            }
            iFile = 1;
        } else {
            iFile = 0;
        }

        /* Init RUNS */
        RUNS      = mxCreateDoubleMatrix(1,2,mxREAL);
        runs      = mxGetPr(RUNS);
        *runs     = 0;
        *(runs+1) = 0;

        /* Get DMCCYCLE */
        CDFlib(SELECT_,ATTR_NAME_,"DMCCYCLE",
                       gENTRY_,0,
                 GET_,gENTRY_DATA_,t,NULL_);
        *record_rate = t[0];

        /* Create time channel name first*/
        varnames[0] = VarNameBuffer[0];
        units[0]    = UnitBuffer[0];
        strcpy(varnames[0],"Time");
        strcpy(units[0],"s");
        nVars = 1;

        /* Read from up to 2 files */
        while (iFile>=0) {

            /* GET UTCTIME */
            CDFlib(SELECT_,CDF_,Id[iFile],
                           ATTR_NAME_,"UTCTIME",
                           gENTRY_,0,
                     GET_,gENTRY_DATA_,&UTCTime,NULL_);
            EPOCHbreakdown(UTCTime,&year,&month,&day,&hour,&minute,&second,&n);
            UTCTime = 3600.0*hour + 60 * minute + second + 0.001*n;

            /* Get BEGCYCLE */
            CDFlib(SELECT_,ATTR_NAME_,"BEGCYCLE",
                           gENTRY_,0,
                     GET_,gENTRY_DATA_,&BegCycle,NULL_);
            MinCycleNr = INT_MAX;
            MaxCycleNr = 0;
```

```
/* Get bit signal names */
CDFlib(SELECT_,zVAR_NAME_,"BIT_SIGNAL_NAME",
         GET_,zVAR_MAXREC_,&n,NULL_);
n++;
if (nVars+n > MAXVAR) {
    while (iFile>=0) {
        CDFlib(SELECT_,CDF_,Id[iFile--],
                CLOSE_,CDF_,NULL_);
    }
    mexErrMsgTxt("cdf_head: Number of variables exceeds program limit");
}
for (i=0 ; i<n ; i++) {
    CDFlib(SELECT_,zVAR_RECNUMBER_,i,
              GET_,zVAR_DATA_,VarNameBuffer[nVars],
                  zVAR_NUMELEMS_,&Len,NULL_);
    VarNameBuffer[nVars][Len] = '\0';
    varnames[nVars] = deblank(VarNameBuffer[nVars]);
    UnitBuffer[nVars][0] = '\0';
    units[nVars] = UnitBuffer[nVars];
    nVars++;
}

/* Get variable names and units */
CDFlib(SELECT_,ATTR_NAME_,"SIGUNIT",
          GET_,CDF_NUMzVARS_,&n,NULL_);
for (i=0 ; i<n ; i++) {
    CDFlib(SELECT_,zVAR_,i,
              GET_,zVAR_NAME_,TextBuffer,
                   zVAR_NUMDIMS_,&NumDims,
                   zVAR_DIMSIZES_,DimSizes,NULL_);
    deblank(TextBuffer);

    /* Cycle channel: Get min/max cycle number for RUNS computing*/
    if (strncmp(TextBuffer,"_CYCLE",6) == 0) {
        CDFlib(SELECT_,zVAR_RECNUMBER_,0,
                  GET_,zVAR_MAXREC_,&Len,
                       zVAR_DATA_,&k,NULL_);
        if (k < MinCycleNr) MinCycleNr = k;
        CDFlib(SELECT_,zVAR_RECNUMBER_,Len,
                  GET_,zVAR_DATA_,&k,NULL_);
        if (k > MaxCycleNr) MaxCycleNr = k;
    }

    /* Bitsignal channel: Skip*/
    else if (strncmp(TextBuffer,"BIT_SIGNAL_",11) == 0) {

    /* Regular channel*/
    } else {
        status = CDFlib(SELECT_,zENTRY_,i,
                           GET_,zENTRY_DATA_,UnitBuffer[nVars],
                                zENTRY_NUMELEMS_,&Len,NULL_);
        if (status != CDF_OK) Len = 0;
        if (Len > UNIT_LEN) Len = UNIT_LEN;
        UnitBuffer[nVars][Len] = '\0';
        deblank(UnitBuffer[nVars]);

        /* Scalar variable or dimensioned variable? */
        if (NumDims == 0) DimSizes[0] = 1;
        if (nVars+DimSizes[0] > MAXVAR) {
            while (iFile>=0) {
                CDFlib(SELECT_,CDF_,Id[iFile--],
                        CLOSE_,CDF_,NULL_);
            }
            mexErrMsgTxt("cdf_head: Number of variables exceeds program limit");
        }
        for (k=0 ; k<DimSizes[0] ; k++) {
            strcpy(VarNameBuffer[nVars],TextBuffer);
            varnames[nVars] = VarNameBuffer[nVars];
            if (NumDims > 0) {
                sprintf(NumStr,"(%d)",k+1);
                strncat(varnames[nVars],NumStr,7);
            }
            units[nVars] = UnitBuffer[nVars-k];
            nVars++;
```

```
                    }
                }
            }

            /* Compute RUNS only once from parent file*/
            if (*runs == *(runs+1)) {
                *runs     = UTCTime + (MinCycleNr - BegCycle) * *record_rate;
                *(runs+1) = UTCTime + (MaxCycleNr - BegCycle) * *record_rate;
            }

            /* Close this CDF */
            CDFlib(CLOSE_,CDF_,NULL_);
            iFile--;
        }

/*****************************************/
/* Get header information from AIRBUS file */
/*****************************************/
    } else if (strcmp(filetype,"AIRBUS") == 0) {

        /* Get RECORD_RATE */
        CDFlib(SELECT_,ATTR_NAME_,"SAMPLING_PERIOD(S)",
                       gENTRY_,0,
                 GET_,gENTRY_DATA_,record_rate,NULL_);

        /* Get RUNS */
        RUNS = mxCreateDoubleMatrix(1,2,mxREAL);
        runs = mxGetPr(RUNS);
        CDFlib(SELECT_,zVAR_NAME_,"GMT",
                  GET_,zVAR_MAXREC_,&n,
                       zVAR_DIMSIZES_,DimSizes,NULL_);
        indices[0] = 0;
        counts[0] = DimSizes[0];
        intervals[0] = 1;

        /* Start time of run*/
        time = mxCalloc(DimSizes[0],sizeof(double));
        CDFlib(SELECT_,zVAR_RECNUMBER_,0,
                       zVAR_RECCOUNT_,1,
                       zVAR_DIMINDICES_,indices,
                       zVAR_DIMCOUNTS_,counts,
                       zVAR_DIMINTERVALS_,intervals,
                  GET_,zVAR_HYPERDATA_,time,NULL_);
        *runs = fmod(time[0],86400);
        tOffset = time[0] - *runs;

        /* End time of run (search for the last valid time value)*/
        CDFlib(SELECT_,zVAR_RECNUMBER_,n,
                  GET_,zVAR_HYPERDATA_,time,NULL_);
        i = DimSizes[0] - 1;
        while ((i > 0) &&
               ((time[i] - tOffset < *runs) || (time[i] - tOffset > *runs + 86400))) {
            i--;
        }
        *(runs+1) = time[i] - tOffset;
        mxFree(time);

        /* Get Variable Names and Units */
        CDFlib(SELECT_,ATTR_NAME_,"UNIT",NULL_);
        for (i=0 ; i<nVars ; i++) {
            CDFlib(SELECT_,zVAR_,i,
                      GET_,zVAR_NAME_,VarNameBuffer[i],NULL_);
            varnames[i] = deblank(VarNameBuffer[i]);
            status = CDFlib(SELECT_,zENTRY_,i,
                                GET_,zENTRY_DATA_,UnitBuffer[i],
                                     zENTRY_NUMELEMS_,&Len,NULL_);
            if (status != CDF_OK) Len = 0;
            if (Len > UNIT_LEN) Len = UNIT_LEN;
            UnitBuffer[i][Len] = '\0';
            deblank(UnitBuffer[i]);
            Len = (long)strlen(UnitBuffer[i]);
            if ((Len > 1) && (UnitBuffer[i][0] == '[') && (UnitBuffer[i][Len-1] == ']')) {
                strcpy(UnitBuffer[i],&UnitBuffer[i][1]);
                UnitBuffer[i][Len-2] = '\0';
```

```
            }
            units[i] = UnitBuffer[i];
        }

/*********************/
/* Unknown file type */
/*********************/
    } else {
        CDFlib(CLOSE_,CDF_,NULL_);
        mexErrMsgTxt("cdf_head: CDF file standard unknown");
    }

    /* Close CDF */
    CDFlib(CLOSE_,CDF_,NULL_);

    /* Interface */
    VARNAMES = mxCreateCharMatrixFromStrings(nVars,(const char**)varnames);
    UNITS    = mxCreateCharMatrixFromStrings(nVars,(const char**)units);
}
```

A.2 cdf_read.c

```
/*
    cdf_read.c  .MEX file implementation for reading
                channel data of a cdf file (IS3, FRG, RCDF, AIRBUS).

    Author: Achim.Jaekel@dlr.de

    The calling syntax is:
    [data,rdt] = cdf_read(filename,varnames,ts,te,rd)

    cdf_read opens the CDF file specified by filename and returns
    channel data.

    INPUT:
    filename - string containing the name of the RCDF file
        varnames - names of channels (default=all)
        ts,te    - start and end time of selected time interval
                   (default=all)
        rd       - factor for data reduction (default=1)

    OUTPUT:
    data     - the data of selected variables
    rdt      - scanning rate of reduced data
*/

/* Includes */
#include <math.h>
#include <string.h>
#include "mex.h"
#include "cdf_sublib.h"

/* Constants*/
#define MAXVAR 5000

/* Arguments */
#define FILENAME  prhs[0]
#define VARNAMES  prhs[1]
#define TS        prhs[2]
#define TE        prhs[3]
#define RD        prhs[4]
#define DATA      plhs[0]
#define RDT       plhs[1]

void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{
    char   filename[CDF_PATHNAME_LEN+1]; /* Pathname of RCDF file */
    char   varnames[MAXVAR][CDF_VAR_NAME_LEN+8]; /* Variable names */
    double ts;                           /* Start time */
    double te;                           /* End time */
    int    rd;                           /* Data reduction */
    double *data;                        /* Channel data */
    double *rdt;                         /* Scanning rate of reduced data */

    int            i,k,iVar,iBit,iFile,MaxFile,Start,End;
    long           mVar,nVar,nBitSignals,nAll;
    int            BegCycle,MinCycle,MaxCycle,LastCycleChannel;
    int            BitSignalSrcId[MAXVAR],BitSignalSrcNr[MAXVAR];
    unsigned int   BitSignalMask[MAXVAR],BitSignalLowBit[MAXVAR];
    char           BitSignalName[MAXVAR][CDF_VAR_NAME_LEN+1];
    char           AllNames[MAXVAR][CDF_VAR_NAME_LEN+1];
    long           NumDims,DimSizes[2],indices[2],counts[2],intervals[2];
    long           year,month,day,hour,minute,second,n,np,npMax,DataType;
    float          fdt;
    double         UTCTime,ddt,tOffset;
    char           *pt;
    char           filetype[8],TextBuffer[1024],NumStr[8];
    CDFid          Id[2];
    CDFstatus      status;
    int            *Cycle;
    double         *DoubleDaten;
    float          *FloatDaten;
```

```
int            *IntDaten;
unsigned int   *UIntDaten;
short          *ShortDaten;
unsigned short *UShortDaten;
char           *CharDaten;
unsigned char  *UCharDaten;
void           *VoidDaten;

/* Initialization*/
strcpy(filetype,"UNKNOWN");
mVar = 0;
ts = -1.0;
te = -2.0;
rd = 1;

/* Check arguments */
if (nrhs < 1 || (!mxIsChar(FILENAME))) {
    mexErrMsgTxt("cdf_read: Filename argument required");
}
if (mxGetN(FILENAME) > CDF_PATHNAME_LEN) {
    mexErrMsgTxt("cdf_read: Filename too long.");
}
mxGetString(FILENAME,filename,CDF_PATHNAME_LEN+1);

if (nrhs > 1 && mxIsChar(VARNAMES)) {
    mVar = (long)mxGetM(VARNAMES);
    nVar = (long)mxGetN(VARNAMES);
    if (mVar > MAXVAR) {
        mexErrMsgTxt("cdf_read: Too many varnames selected");
    }
    CharDaten = mxCalloc(mVar*nVar+1,sizeof(mxChar));
    mxGetString(VARNAMES,CharDaten,mVar*nVar+1);
    pt = CharDaten;
    for (k=0 ; k<nVar ; k++) {
        for (i=0 ; i<mVar ; i++) {
            varnames[i][k] = *pt;
            pt++;
        }
    }
    for (i=0; i<mVar; i++) {
        varnames[i][nVar] = '\0';
        deblank(varnames[i]);
    }
    mxFree(CharDaten);
}

if (nrhs > 3 && mxIsDouble(TS) && mxIsDouble(TE)) {
    ts = mxGetScalar(TS);
    te = mxGetScalar(TE);
}

if (nrhs > 4 && mxIsNumeric(RD)) {
    rd = (int)(mxGetScalar(RD)+0.5);
}

/* Open file */
status = CDFlib(  OPEN_,CDF_,filename,&Id[0],
                SELECT_,CDF_READONLY_MODE_,READONLYon,NULL_);
if (status != CDF_OK) disperr(status);

/* Get TYPE of the file (DLR-FT standards)*/
status = CDFlib(SELECT_,ATTR_NAME_,"TYPE",
                        gENTRY_,0,
                   GET_,gENTRY_DATA_,TextBuffer,NULL_);
if (status == CDF_OK) {
    if (strncmp(TextBuffer,"IS3",3) == 0) strcpy(filetype,"IS3");
    if (strncmp(TextBuffer,"FRG",3) == 0) strcpy(filetype,"FRG");
    if (strncmp(TextBuffer,"RCDF",4) == 0) strcpy(filetype,"RCDF");

/* AIRBUS standard?*/
} else {
    status = CDFlib(CONFIRM_,ATTR_EXISTENCE_,"SAMPLING_PERIOD(S)",NULL_);
    if (status == CDF_OK) strcpy(filetype,"AIRBUS");
}
```

```
    /* Interface */
    RDT = mxCreateDoubleMatrix(1,1,mxREAL);
    rdt = mxGetPr(RDT);

/*****************************/
/* Read data from IS3/FRG-file */
/*****************************/
    if ((strcmp(filetype,"IS3") == 0) || (strcmp(filetype,"FRG") == 0)) {

        /* Read RECORD_RATE */
        CDFlib(SELECT_,ATTR_NAME_,"RECORD_RATE",
                       gENTRY_,0,
                 GET_,gENTRY_DATA_,&fdt,NULL_);

        /* Get SCANNING only for FRG files*/
        if (strcmp(filetype,"FRG") == 0) {
            CDFlib(SELECT_,ATTR_NAME_,"SCANNING",
                           gENTRY_,0,
                     GET_,gENTRY_DATA_,TextBuffer,NULL_);
        } else {
            strcpy(TextBuffer,"LIN");
        }
        if (strncmp(TextBuffer,"LOG",3) == 0) {
            *rdt = pow(fdt,rd);
        } else {
            *rdt = fdt*rd;
        }

        /* Select all varnames, if none are specified */
        if (mVar == 0) {
            CDFlib(GET_,CDF_NUMzVARS_,&mVar,NULL_);
            if (mVar > MAXVAR) {
                CDFlib(CLOSE_,CDF_,NULL_);
                mexErrMsgTxt("cdf_read: Number of variables exceeds program limit");
            }
            for (iVar=0 ; iVar<mVar ; iVar++) {
                CDFlib(SELECT_,zVAR_,iVar,
                           GET_,zVAR_NAME_,varnames[iVar],NULL_);
                deblank(varnames[iVar]);
            }
        }

        /* Read time/frequency channel */
        CDFlib(SELECT_,zVAR_,0,
                  GET_,zVAR_MAXREC_,&np,NULL_);
        np++;
        FloatDaten = mxCalloc(np,2*sizeof(float));
        indices[0]   = 0;
        indices[1]   = 0;
        counts[0]    = 1;
        counts[1]    = 1;
        intervals[0] = 1;
        intervals[1] = 1;
        CDFlib(SELECT_,zVAR_RECNUMBER_,0,
                       zVAR_RECCOUNT_,np,
                       zVAR_DIMINDICES_,indices,
                       zVAR_DIMCOUNTS_,counts,
                       zVAR_DIMINTERVALS_,intervals,
                       GET_,zVAR_HYPERDATA_,FloatDaten,NULL_);

        /* Find start and end indices */
        if (ts > te) {
            Start = 0;
            End   = np-1;
        } else {
            ddt = 0.5 * fdt;
            for (Start=0 ; Start<np-1 ; Start++) {
                if (ts < FloatDaten[Start]+ddt) break;
            }
            for (End=Start ; End<np-1 ; End++) {
                if (te < FloatDaten[End]+ddt) break;
            }
        }
```

```
        /* Compute new np */
        np = (End - Start) / rd + 1;

        /* Allocate memory for interface data */
        DATA = mxCreateDoubleMatrix(np,counts[0]*mVar,mxREAL);
        data = mxGetPr(DATA);
        indices[0]   = 0;
        indices[1]   = 0;
        counts[0]    = 1;
        counts[1]    = 1;
        intervals[0] = 1;
        intervals[1] = 1;
        if (strcmp(filetype,"FRG") == 0) counts[0] = 2;

        /* Read data*/
        for (iVar=0 ; iVar<mVar ; iVar++) {
            status = CDFlib(SELECT_,zVAR_NAME_,varnames[iVar],
                                    zVAR_RECNUMBER_,Start,
                                    zVAR_RECCOUNT_,np,
                                    zVAR_RECINTERVAL_,rd,
                                    zVAR_DIMINDICES_,indices,
                                    zVAR_DIMCOUNTS_,counts,
                                    zVAR_DIMINTERVALS_,intervals,
                               GET_,zVAR_HYPERDATA_,FloatDaten,NULL_);
            if (status != CDF_OK) {
                mxFree(FloatDaten);
                CDFlib(CLOSE_,CDF_,NULL_);
                printf("zVAR_NAME = %s\n",varnames[iVar]);
                disperr(status);
            }

            /* Typecast data to double*/
            for (k=0 ; k<np ; k++) {
                *data = FloatDaten[counts[0]*k];
                data++;
            }
            if (strcmp(filetype,"FRG") == 0) {
                for (k=0 ; k<np ; k++) {
                    *data = FloatDaten[2*k+1];
                    data++;
                }
            }
        }

        /* Free memory */
        mxFree(FloatDaten);

/***************************/
/* Read data from RCDF-file */
/***************************/
    } else if (strcmp(filetype,"RCDF") == 0) {

        /* Get PARENT_FILENAME */
        status = CDFlib(CONFIRM_,ATTR_EXISTENCE_,"PARENT_FILENAME",NULL_);
        if (status == CDF_OK) {
            pt = strrchr(filename,filesep());
            if (pt == NULL) {
                pt = filename;
            } else {
                pt++;
            }
            CDFlib(SELECT_,ATTR_NAME_,"PARENT_FILENAME",
                           gENTRY_,0,
                      GET_,gENTRY_DATA_,pt,
                           gENTRY_NUMELEMS_,&n,NULL_);
            *(pt+n) = '\0';

            /* Open parent file */
            status = CDFlib(  OPEN_,CDF_,filename,&Id[1],
                            SELECT_,CDF_READONLY_MODE_,READONLYon,NULL_);
            if (status != CDF_OK) {
                CDFlib(SELECT_,CDF_,Id[0],
                         CLOSE_,CDF_,NULL_);
```

```
            printf("PARENT_FILENAME = %s\n",filename);
            disperr(status);
        }
        MaxFile = 1;
    } else {
        MaxFile = 0;
    }

    /* Select all varnames, if none are specified */
    if (mVar == 0) {
        strcpy(varnames[0],"Time");
        mVar = 1;

        /* Do it for 2 files, if file is a parentfile <=> childfile combination */
        for (iFile=MaxFile ; iFile>=0 ; iFile--) {
            CDFlib(SELECT_,CDF_,Id[iFile],NULL_);

            /* Select all bit signals */
            CDFlib(SELECT_,zVAR_NAME_,"BIT_SIGNAL_NAME",
                      GET_,zVAR_MAXREC_,&n,NULL_);
            n++;
            if (mVar + n > MAXVAR) {
                for (i=0 ; i<=MaxFile ; i++) {
                    CDFlib(SELECT_,CDF_,Id[i],
                            CLOSE_,CDF_,NULL_);
                }
                mexErrMsgTxt("cdf_read: Number of variables exceeds program limit");
            }
            for (i=0 ; i<n ; i++) {
                CDFlib(SELECT_,zVAR_RECNUMBER_,i,
                               zVAR_RECCOUNT_,1,
                          GET_,zVAR_DATA_,varnames[mVar],
                               zVAR_NUMELEMS_,&np,NULL_);
                varnames[mVar][np] = '\0';
                deblank(varnames[mVar++]);
            }

            /* Select all other variables */
            CDFlib(GET_,CDF_NUMzVARS_,&n,NULL_);
            for (iVar=0 ; iVar<n ; iVar++) {
                CDFlib(SELECT_,zVAR_,iVar,
                          GET_,zVAR_NAME_,TextBuffer,
                               zVAR_NUMDIMS_,&NumDims,
                               zVAR_DIMSIZES_,DimSizes,NULL_);
                deblank(TextBuffer);
                if ((strncmp(TextBuffer,"_CYCLE",6) != 0) &&
                    (strncmp(TextBuffer,"BIT_SIGNAL_",11) != 0)) {

                    /* Scalar variable or dimensioned variable? */
                    if (NumDims == 0) DimSizes[0] = 1;
                    if (mVar+DimSizes[0] > MAXVAR) {
                        for (i=0; i<=MaxFile; i++) {
                            CDFlib(SELECT_,CDF_,Id[i],
                                    CLOSE_,CDF_,NULL_);
                        }
                      mexErrMsgTxt("cdf_read: Number of variables exceeds program limit");
                    }
                    for (k=0 ; k<DimSizes[0] ; k++) {
                        strcpy(varnames[mVar],TextBuffer);
                        if (DimSizes[0] > 1) {
                            sprintf(NumStr,"(%d)",k+1);
                            strncat(varnames[mVar],NumStr,7);
                        }
                        mVar++;
                    }
                }
            }
        }
    }

    /* Read record rate from parent file*/
    CDFlib(SELECT_,CDF_,Id[MaxFile],NULL_);
    CDFlib(SELECT_,ATTR_NAME_,"DMCCYCLE",
                  gENTRY_,0,
```

```
          GET_,gENTRY_DATA_,&fdt,NULL_);
*rdt = fdt*rd;

/* Do it for up to 2 files (parent <=> child combination */
for (iFile=MaxFile ; iFile>=0 ; iFile--) {

    /* Read attributes*/
    MinCycle         = INT_MAX;
    MaxCycle         = 0;
    LastCycleChannel = -1;
    CDFlib(SELECT_,CDF_,Id[iFile],NULL_);
    CDFlib(SELECT_,ATTR_NAME_,"BEGCYCLE",
                   gENTRY_,0,
              GET_,gENTRY_DATA_,&BegCycle,NULL_);
    CDFlib(SELECT_,ATTR_NAME_,"UTCTIME",
                   gENTRY_,0,
              GET_,gENTRY_DATA_,&UTCTime,NULL_);
    EPOCHbreakdown(UTCTime,&year,&month,&day,&hour,&minute,&second,&n);
    UTCTime = 3600.0*hour + 60 * minute + second + 0.001*n;

    /* Read names, sourceids and masks of all bit signals */
    CDFlib(SELECT_,zVAR_NAME_,"BIT_SIGNAL_NAME",
              GET_,zVAR_MAXREC_,&nBitSignals,NULL_);
    nBitSignals++;
    if (nBitSignals > MAXVAR) {
        for (i=0; i<=iFile; i++) {
            CDFlib(SELECT_,CDF_,Id[i],
                     CLOSE_,CDF_,NULL_);
        }
        mexErrMsgTxt("cdf_read: Number of bitsignals exceeds program limit");
    }
    for (i=0; i<nBitSignals; i++) {
        CDFlib(SELECT_,zVAR_RECNUMBER_,i,
                  GET_,zVAR_DATA_,BitSignalName[i],NULL_);
        BitSignalName[i][CDF_VAR_NAME_LEN] = '\0';
        deblank(BitSignalName[i]);
    }
    CDFlib(SELECT_,zVAR_NAME_,"BIT_SIGNAL_SRCID",
                   zVAR_RECNUMBER_,0,
                   zVAR_RECCOUNT_,nBitSignals,
                   zVAR_RECINTERVAL_,1,
              GET_,zVAR_HYPERDATA_,BitSignalSrcId,NULL_);
    CDFlib(SELECT_,zVAR_NAME_,"BIT_SIGNAL_MASK",
                   zVAR_RECNUMBER_,0,
                   zVAR_RECCOUNT_,nBitSignals,
                   zVAR_RECINTERVAL_,1,
              GET_,zVAR_HYPERDATA_,BitSignalMask,NULL_);
    for (i=0; i<nBitSignals; i++) {
        BitSignalLowBit[i]=0;
        if (BitSignalMask[i] != 0) {
            while ((BitSignalMask[i] & (1 << BitSignalLowBit[i])) == 0) {
                BitSignalLowBit[i]++;
            }
        }
    }

    /* Scan all zVariables*/
    npMax = 0;
    CDFlib(SELECT_,ATTR_NAME_,"SIGNALID",
              GET_,CDF_NUMzVARS_,&nAll,NULL_);
    if (nAll > MAXVAR) {
        for (i=0; i<=MaxFile; i++) {
            CDFlib(SELECT_,CDF_,Id[i],
                     CLOSE_,CDF_,NULL_);
        }
        mexErrMsgTxt("cdf_read: Number of variables exceeds program limit");
    }
    for (iVar=0; iVar<nAll; iVar++) {
        CDFlib(SELECT_,zVAR_,iVar,
                  GET_,zVAR_NAME_,AllNames[iVar],
                       zVAR_MAXREC_,&np,NULL_);
        deblank(AllNames[iVar]);

        /* Find min/max cycle numbers of file */
```

```
    if (strncmp(AllNames[iVar],"_CYCLE",6) == 0) {
        CDFlib(SELECT_,zVAR_RECNUMBER_,0,
                  GET_,zVAR_DATA_,&k,NULL_);
        if (k < MinCycle) MinCycle = k;
        CDFlib(SELECT_,zVAR_RECNUMBER_,np,
                  GET_,zVAR_DATA_,&k,NULL_);
        if (k > MaxCycle) MaxCycle = k;

        /* Get maximum number of records of all variables*/
        if (np >= npMax) npMax = np+1;
    }

    /* Scan all bitsignals, if this zVariable is a bitsignals source */
    else if (strncmp(AllNames[iVar],"BIT_SIGNAL_",11) != 0) {
        CDFlib(SELECT_,zENTRY_,iVar,
                  GET_,zENTRY_DATA_,&k,NULL_);
        for (i=0 ; i<nBitSignals ; i++) {
            if (k == BitSignalSrcId[i]) {
                BitSignalSrcNr[i] = iVar;
            }
        }
    }
}

/* Find start and end cycles */
if (ts > te) {
    Start = MinCycle;
    End   = MaxCycle;
    ts    = UTCTime + (Start-BegCycle) * fdt;
    te    = UTCTime + (End-BegCycle) * fdt;
} else {
    Start = BegCycle + (int)((ts - UTCTime) / fdt + 0.5);
    End   = Start + (int)((te-ts) / fdt + 0.5);
}

/* Allocate memory for interface data */
if (iFile == MaxFile) {
    nVar = (End - Start) / rd + 1;
    DATA = mxCreateDoubleMatrix(nVar,mVar,mxREAL);
}

/* Allocate memory for temporary data to read*/
Cycle       = mxCalloc(npMax,sizeof(int));
VoidDaten   = mxCalloc(npMax,sizeof(double));
DoubleDaten = (double*)VoidDaten;
FloatDaten  = (float*)VoidDaten;
IntDaten    = (int*)VoidDaten;
UIntDaten   = (unsigned int*)VoidDaten;
ShortDaten  = (short*)VoidDaten;
UShortDaten = (unsigned short*)VoidDaten;
CharDaten   = (char*)VoidDaten;
UCharDaten  = (unsigned char*)VoidDaten;

/* Search for all selected variable names */
indices[1]   = 0;
counts[0]    = 1;
counts[1]    = 1;
intervals[0] = 1;
intervals[1] = 1;
CDFlib(SELECT_,ATTR_NAME_,"CYCLECHN",NULL_);
for (iVar=0 ; iVar<mVar ; iVar++) {

    /* Locate data pointer to first entry of variables column */
    data = mxGetPr(DATA) + iVar * nVar;

    /* Compute time channel only once */
    if ((iFile == MaxFile) && (strncmp(varnames[iVar],"Time",4) == 0)) {
        for (i=Start ; i<=End ; i+=rd) {
            *data = UTCTime + (i-BegCycle) * fdt;
            data++;
        }
        continue;
    }
```

```
/* Get name and dimension of channel */
pt = strchr(varnames[iVar],'(');
if (pt != NULL) {
    sscanf(pt,"(%d)",&indices[0]);
    indices[0]--;
    *pt = '\0';
} else {
    indices[0] = 0;
}

/* Select source channel of bit signal by number*/
for (iBit=0; iBit<nBitSignals; iBit++) {
    if (strcmp(varnames[iVar],BitSignalName[iBit]) == 0) {
        CDFlib(SELECT_,zVAR_,BitSignalSrcNr[iBit],
                        zENTRY_,BitSignalSrcNr[iBit],NULL_);
        break;
    }
}

/* Select channel for non bit signal by number*/
if (iBit >= nBitSignals) {
    for (i=0; i<nAll; i++) {
        if (strcmp(varnames[iVar],AllNames[i]) == 0) {
            CDFlib(SELECT_,zVAR_,i,
                           zENTRY_,i,NULL_);
            break;
        }
    }
    if (i >= nAll) {
        continue;
    }
}

/* Read data channel */
CDFlib(GET_,zVAR_MAXREC_,&np,NULL_);
np++;
CDFlib(SELECT_,zVAR_RECNUMBER_,0,
               zVAR_RECCOUNT_,np,
               zVAR_RECINTERVAL_,1,
               zVAR_DIMINDICES_,indices,
               zVAR_DIMCOUNTS_,counts,
               zVAR_DIMINTERVALS_,intervals,
         GET_,zVAR_DATATYPE_,&DataType,
               zVAR_HYPERDATA_,VoidDaten,NULL_);

/* Get cycle channel number*/
CDFlib(GET_,zENTRY_DATA_,&k,NULL_);

/* Read cycle channel only once*/
if ((LastCycleChannel < 0) || (k != LastCycleChannel)) {
    LastCycleChannel = k;
    sprintf(TextBuffer,"_CYCLE%d",LastCycleChannel);
    for (i=0; i<nAll; i++) {
        if (strcmp(TextBuffer,AllNames[i]) == 0) {
            CDFlib(SELECT_,zVAR_,i,
                           zVAR_RECNUMBER_,0,
                           zVAR_RECCOUNT_,np,
                           zVAR_RECINTERVAL_,1,
                      GET_,zVAR_HYPERDATA_,Cycle,NULL_);
            break;
        }
    }
}

/* Evaluate bit signal */
if (iBit < nBitSignals) {
    switch (DataType) {
        case CDF_INT4:
        case CDF_UINT4:
            for (i=0 ; i<np ; i++) {
                IntDaten[i] = (IntDaten[i] & BitSignalMask[iBit])
                            >> BitSignalLowBit[iBit];
            }
            break;
```

```
                        case CDF_INT2:
                        case CDF_UINT2:
                            for (i=0 ; i<np ; i++) {
                                ShortDaten[i] = (ShortDaten[i] & BitSignalMask[iBit])
                                            >> BitSignalLowBit[iBit];
                            }
                            break;
                        case CDF_INT1:
                        case CDF_UINT1:
                        case CDF_BYTE:
                        case CDF_CHAR:
                        case CDF_UCHAR:
                            for (i=0 ; i<np ; i++) {
                                CharDaten[i] = (CharDaten[i] & BitSignalMask[iBit])
                                            >> BitSignalLowBit[iBit];
                            }
                            break;
                    }
                }

                /* Interpolate data and typecast to double*/
                k = 0;
                switch (DataType) {
                    case CDF_REAL8:
                    case CDF_DOUBLE:
                    case CDF_EPOCH:
                        for (i=Start ; i<=End ; i+=rd) {
gDOUBLE: /* Linear interpolation*/
                            if (i <= Cycle[0]) {
                                *data++ = DoubleDaten[0];
                            } else if (i >= Cycle[np-1]) {
                                *data++ = DoubleDaten[np-1];
                            } else if (i < Cycle[k]) {
                                ddt = (DoubleDaten[k]-DoubleDaten[k-1]) /
                                      (Cycle[k]-Cycle[k-1]);
                                *data++ = DoubleDaten[k-1] + ddt*(i-Cycle[k-1]);
                            } else {
                                k++;
                                goto gDOUBLE;
                            }
                        }
                        break;

                    case CDF_REAL4:
                    case CDF_FLOAT:
                        for (i=Start ; i<=End ; i+=rd) {
gFLOAT: /* Linear interpolation*/
                            if (i <= Cycle[0]) {
                                *data++ = FloatDaten[0];
                            } else if (i >= Cycle[np-1]) {
                                *data++ = FloatDaten[np-1];
                            } else if (i < Cycle[k]) {
                                ddt = (FloatDaten[k]-FloatDaten[k-1]) /
                                      (Cycle[k]-Cycle[k-1]);
                                *data++ = FloatDaten[k-1] + ddt*(i-Cycle[k-1]);
                            } else {
                                k++;
                                goto gFLOAT;
                            }
                        }
                        break;

                    case CDF_INT4:
                        for (i=Start ; i<=End ; i+=rd) {
gLONG: /* Sample and hold */
                            if (i <= Cycle[0]) {
                                *data++ = IntDaten[0];
                            } else if (i >= Cycle[np-1]) {
                                *data++ = IntDaten[np-1];
                            } else if (i < Cycle[k]) {
                                *data++ = IntDaten[k-1];
                            } else {
                                k++;
                                goto gLONG;
```

```
                        }
                    }
                    break;

                case CDF_UINT4:
                    for (i=Start ; i<=End ; i+=rd) {
gULONG: /* Sample and hold */
                        if (i <= Cycle[0]) {
                            *data++ = UIntDaten[0];
                        } else if (i >= Cycle[np-1]) {
                            *data++ = UIntDaten[np-1];
                        } else if (i < Cycle[k]) {
                            *data++ = UIntDaten[k-1];
                        } else {
                            k++;
                            goto gULONG;
                        }
                    }
                    break;

                case CDF_INT2:
                    for (i=Start ; i<=End ; i+=rd) {
gSHORT: /* Sample and hold */
                        if (i <= Cycle[0]) {
                            *data++ = ShortDaten[0];
                        } else if (i >= Cycle[np-1]) {
                            *data++ = ShortDaten[np-1];
                        } else if (i < Cycle[k]) {
                            *data++ = ShortDaten[k-1];
                        } else {
                            k++;
                            goto gSHORT;
                        }
                    }
                    break;

                case CDF_UINT2:
                    for (i=Start ; i<=End ; i+=rd) {
gUSHORT: /* Sample and hold */
                        if (i <= Cycle[0]) {
                            *data++ = UShortDaten[0];
                        } else if (i >= Cycle[np-1]) {
                            *data++ = UShortDaten[np-1];
                        } else if (i < Cycle[k]) {
                            *data++ = UShortDaten[k-1];
                        } else {
                            k++;
                            goto gUSHORT;
                        }
                    }
                    break;

                case CDF_INT1:
                case CDF_BYTE:
                case CDF_CHAR:
                    for (i=Start ; i<=End ; i+=rd) {
gCHAR: /* Sample and hold */
                        if (i <= Cycle[0]) {
                            *data++ = CharDaten[0];
                        } else if (i >= Cycle[np-1]) {
                            *data++ = CharDaten[np-1];
                        } else if (i < Cycle[k]) {
                            *data++ = CharDaten[k-1];
                        } else {
                            k++;
                            goto gCHAR;
                        }
                    }
                    break;

                case CDF_UINT1:
                case CDF_UCHAR:
                    for (i=Start ; i<=End ; i+=rd) {
gUCHAR: /* Sample and hold */
```

```
                            if (i <= Cycle[0]) {
                                *data++ = UCharDaten[0];
                            } else if (i >= Cycle[np-1]) {
                                *data++ = UCharDaten[np-1];
                            } else if (i < Cycle[k]) {
                                *data++ = UCharDaten[k-1];
                            } else {
                                k++;
                                goto gUCHAR;
                            }
                        }
                        break;
                }
            }

            /* Free memory*/
            mxFree(Cycle);
            mxFree(VoidDaten);

            /* Close this cdf file*/
            CDFlib(CLOSE_,CDF_,NULL_);
        }

/*****************************/
/* Read data from AIRBUS file */
/*****************************/
    } else if (strcmp(filetype,"AIRBUS") == 0) {

        /* Get RECORD_RATE */
        CDFlib(SELECT_,ATTR_NAME_,"SAMPLING_PERIOD(S)",
                       gENTRY_,0,
                 GET_,gENTRY_DATA_,&ddt,NULL_);
        *rdt = ddt*rd;

        /* Select all varnames, if none are specified */
        if (mVar == 0) {
            CDFlib(GET_,CDF_NUMzVARS_,&mVar,NULL_);
            if (mVar > MAXVAR) {
                CDFlib(CLOSE_,CDF_,NULL_);
                mexErrMsgTxt("cdf_read: Number of variables exceeds program limit");
            }
            for (iVar=0 ; iVar<mVar ; iVar++) {
                CDFlib(SELECT_,zVAR_,iVar,
                           GET_,zVAR_NAME_,varnames[iVar],NULL_);
                deblank(varnames[iVar]);
            }
        }

        /* Allocate memory for reading GMT and data channel*/
        CDFlib(SELECT_,zVAR_NAME_,"GMT",
                  GET_,zVAR_MAXREC_,&np,
                       zVAR_DIMSIZES_,DimSizes,NULL_);
        np++;
        npMax       = np * DimSizes[0];
        VoidDaten   = mxCalloc(npMax,sizeof(double));
        DoubleDaten = (double*)VoidDaten;
        FloatDaten  = (float*)VoidDaten;
        IntDaten    = (int*)VoidDaten;
        UIntDaten   = (unsigned int*)VoidDaten;
        ShortDaten  = (short*)VoidDaten;
        UShortDaten = (unsigned short*)VoidDaten;
        CharDaten   = (char*)VoidDaten;
        UCharDaten  = (unsigned char*)VoidDaten;

        /* Read GMT*/
        indices[0]   = 0;
        counts[0]    = DimSizes[0];
        intervals[0] = 1;
        CDFlib(SELECT_,zVAR_RECNUMBER_,0,
                       zVAR_RECCOUNT_,np,
                       zVAR_RECINTERVAL_,1,
                       zVAR_DIMINDICES_,indices,
                       zVAR_DIMCOUNTS_,counts,
                       zVAR_DIMINTERVALS_,intervals,
```

```
          GET_,zVAR_HYPERDATA_,VoidDaten,NULL_);

/* Remove time offset from begin of the year until today */
tOffset = DoubleDaten[0] - fmod(DoubleDaten[0],86400);
for (i=0; i<npMax; i++) {
    DoubleDaten[i] = DoubleDaten[i] - tOffset;
}

/* Search the last valid time value*/
while ((npMax > 1) &&
       ((DoubleDaten[npMax-1] < 0.0) || (DoubleDaten[npMax-1] > 86400))) {
    npMax--;
}

/* Find start and end indices */
if (ts > te) {
    Start = 0;
    End   = npMax-1;
} else {
    ddt = 0.5 * ddt;
    for (Start=0 ; Start<npMax-1 ; Start++) {
        if (ts < DoubleDaten[Start]+ddt) break;
    }
    for (End=Start ; End<npMax-1 ; End++) {
        if (te < DoubleDaten[End]+ddt) break;
    }
}

/* Allocate memory for interface data */
n = (End - Start) / rd + 1;
DATA = mxCreateDoubleMatrix(n,mVar,mxREAL);
data = mxGetPr(DATA);

/* Read data*/
for (iVar=0 ; iVar<mVar ; iVar++) {
    status = CDFlib(SELECT_,zVAR_NAME_,varnames[iVar],
                            zVAR_RECNUMBER_,0,
                            zVAR_RECCOUNT_,np,
                            zVAR_RECINTERVAL_,1,
                            zVAR_DIMINDICES_,indices,
                            zVAR_DIMCOUNTS_,counts,
                            zVAR_DIMINTERVALS_,intervals,
                       GET_,zVAR_DATATYPE_,&DataType,
                            zVAR_HYPERDATA_,VoidDaten,NULL_);
    if (status != CDF_OK) {
        mxFree(VoidDaten);
        CDFlib(CLOSE_,CDF_,NULL_);
        printf("zVAR_NAME = %s\n",varnames[iVar]);
        disperr(status);
    }

    /* GMT: Remove time offset from begin of the year until today */
    if (strncmp(varnames[iVar],"GMT",3) == 0) {
        for (i=0; i<npMax; i++) {
            DoubleDaten[i] = DoubleDaten[i] - tOffset;
        }
    }

    /* Copy data*/
    switch (DataType) {
        case CDF_REAL8:
        case CDF_DOUBLE:
        case CDF_EPOCH:
            for (i=Start ; i<=End ; i+=rd) {
                *data++ = DoubleDaten[i];
            }
            break;
        case CDF_REAL4:
        case CDF_FLOAT:
            for (i=Start ; i<=End ; i+=rd) {
                *data++ = FloatDaten[i];
            }
            break;
        case CDF_INT4:
```

```
                    for (i=Start ; i<=End ; i+=rd) {
                        *data++ = IntDaten[i];
                    }
                    break;
                case CDF_UINT4:
                    for (i=Start ; i<=End ; i+=rd) {
                        *data++ = UIntDaten[i];
                    }
                    break;
                case CDF_INT2:
                    for (i=Start ; i<=End ; i+=rd) {
                        *data++ = ShortDaten[i];
                    }
                    break;
                case CDF_UINT2:
                    for (i=Start ; i<=End ; i+=rd) {
                        *data++ = UShortDaten[i];
                    }
                    break;
                case CDF_INT1:
                case CDF_BYTE:
                case CDF_CHAR:
                    for (i=Start ; i<=End ; i+=rd) {
                        *data++ = CharDaten[i];
                    }
                    break;
                case CDF_UINT1:
                case CDF_UCHAR:
                    for (i=Start ; i<=End ; i+=rd) {
                        *data++ = UCharDaten[i];
                    }
                    break;
            }
        }

        /* Free memory*/
        mxFree(VoidDaten);

/*********************/
/* Unknown file type */
/*********************/
    } else {
        CDFlib(CLOSE_,CDF_,NULL_);
        mexErrMsgTxt("cdf_read: CDF file standard unknown");
    }

    /* Close file */
    CDFlib(CLOSE_,CDF_,NULL_);
}
```

A.3 cdf_hardread.c

```
/*
    cdf_hardread.c  .MEX file implementation for reading
                    channel data of a RCDF file without interpolation of
                    missing values.

    Author: Achim.Jaekel@dlr.de

    The calling syntax is:
    [data,intdata] = cdf_hardread(filename,varname,ts,te)

    cdf_hardread opens the CDF file specified by filename and
    returns channel data.

    INPUT:
      filename  - string containing the name of the RCDF file
      varname   - string containing the channel name
          ts,te - start and end time of selected time interval (default=all)

    OUTPUT:
      data     - the data of the time and the selected channel
      intdata  - 1=Integer Datatype, 0=Double Datatype
*/

/* Includes */
#include <math.h>
#include <string.h>
#include "mex.h"
#include "cdf_sublib.h"

/* Arguments */
#define FILENAME prhs[0]
#define VARNAME  prhs[1]
#define TS       prhs[2]
#define TE       prhs[3]
#define DATA     plhs[0]
#define INTDATA  plhs[1]

void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{
    char   filename[CDF_PATHNAME_LEN+1]; /* Pathname of RCDF file */
    char   varname[CDF_VAR_NAME_LEN+8];  /* Variable name */
    double ts;                           /* Start time */
    double te;                           /* End time */
    double *data;                        /* Time and Channel data */
    double *intdata;                     /* 0=Double Channel, 1=Integer Channel */

    int            i,k,iFile,iVar;
    int            Start,End,BegCycle;
    bool           isBitSignal;
    int            BitSignalSrcId;
    unsigned int   BitSignalMask,BitSignalLowBit;
    long           indices[2],counts[2],intervals[2];
    long           year,month,day,hour,minute,second,n,np,DataType;
    float          dt;
    double         UTCTime,eps;
    char           *pt;
    char           TextBuffer[1024];
    CDFid          Id[2];
    CDFstatus      status;
    int            *Cycle;
    double         *DoubleDaten;
    float          *FloatDaten;
    int            *IntDaten;
    unsigned int   *UIntDaten;
    short          *ShortDaten;
    unsigned short *UShortDaten;
    char           *CharDaten;
    unsigned char  *UCharDaten;
    void           *VoidDaten;
```

```
/* Initialization*/
ts = -1.0;
te = -2.0;

/* Check arguments */
if (nrhs < 1 || (!mxIsChar(FILENAME))) {
    mexErrMsgTxt("cdf_hardread: Filename argument required");
}
if (mxGetN(FILENAME) > CDF_PATHNAME_LEN) {
    mexErrMsgTxt("cdf_hardread: Filename too long");
}
mxGetString(FILENAME,filename,CDF_PATHNAME_LEN+1);

if (nrhs < 2 || (!mxIsChar(VARNAME))) {
    mexErrMsgTxt("cdf_hardread: Varname argument required");
}
if (mxGetN(VARNAME) > CDF_VAR_NAME_LEN) {
    mexErrMsgTxt("cdf_hardread: Varname too long");
}
mxGetString(VARNAME,varname,CDF_VAR_NAME_LEN+1);
deblank(varname);

if (nrhs > 3 && mxIsDouble(TS) && mxIsDouble(TE)) {
    ts = mxGetScalar(TS);
    te = mxGetScalar(TE);
}

/* Interface */
INTDATA  = mxCreateDoubleMatrix(1, 1, mxREAL);
intdata  = mxGetPr(INTDATA);
*intdata = 0;

/* Open file */
status = CDFlib(  OPEN_,CDF_,filename,&Id[0],
                SELECT_,CDF_READONLY_MODE_,READONLYon,
                        ATTR_NAME_,"TYPE",
                        gENTRY_,0,
                   GET_,gENTRY_DATA_,TextBuffer,NULL_);
if (status != CDF_OK) {
        CDFlib(SELECT_,CDF_,Id[0],
                CLOSE_,CDF_,NULL_);
    disperr(status);
}

/* Read data from RCDF-file */
if (strncmp(TextBuffer,"RCDF",4) == 0) {

    /* Get PARENT_FILENAME */
    status = CDFlib(CONFIRM_,ATTR_EXISTENCE_,"PARENT_FILENAME",NULL_);
    if (status == CDF_OK) {
        pt = strrchr(filename,filesep());
        if (pt == NULL) {
            pt = filename;
        } else {
            pt++;
        }
        CDFlib(SELECT_,ATTR_NAME_,"PARENT_FILENAME",
                       gENTRY_,0,
                  GET_,gENTRY_DATA_,pt,
                       gENTRY_NUMELEMS_,&n,NULL_);
        *(pt+n) = '\0';

        /* Open parent file */
        status = CDFlib(  OPEN_,CDF_,filename,&Id[1],
                        SELECT_,CDF_READONLY_MODE_,READONLYon,NULL_);
        if (status != CDF_OK) {
            CDFlib(SELECT_,CDF_,Id[0],
                    CLOSE_,CDF_,NULL_);
            printf("PARENT_FILENAME = %s\n",filename);
            disperr(status);
        }
        iFile = 1;
    } else {
        iFile = 0;
```

```
}

/* Read DMCCYCLE */
CDFlib(SELECT_,ATTR_NAME_,"DMCCYCLE",
               gENTRY_,0,
          GET_,gENTRY_DATA_,&dt,NULL_);

/* Do it for up to 2 files (parent <=> child combination */
while (iFile>=0) {
    CDFlib(SELECT_,CDF_,Id[iFile],NULL_);

    /* Read BEGCYCLE */
    CDFlib(SELECT_,ATTR_NAME_,"BEGCYCLE",
                   gENTRY_,0,
              GET_,gENTRY_DATA_,&BegCycle,NULL_);

    /* Read UTCTIME */
    CDFlib(SELECT_,ATTR_NAME_,"UTCTIME",
                   gENTRY_,0,
              GET_,gENTRY_DATA_,&UTCTime,NULL_);
    EPOCHbreakdown(UTCTime,&year,&month,&day,&hour,&minute,&second,&n);
    UTCTime = 3600.0*hour + 60 * minute + second + 0.001*n;

    /* Read names, sourceids and masks of bit signals */
    isBitSignal = false;
    CDFlib(SELECT_,zVAR_NAME_,"BIT_SIGNAL_NAME",
              GET_,zVAR_MAXREC_,&n,NULL_);
    n++;
    for (i=0 ; i<n ; i++) {
        CDFlib(SELECT_,zVAR_RECNUMBER_,i,
                  GET_,zVAR_DATA_,TextBuffer,NULL_);
        TextBuffer[CDF_VAR_NAME_LEN] = '\0';
        deblank(TextBuffer);

        /* Is the selected varname this bitsignal?*/
        if (strcmp(varname,TextBuffer) == 0) {
            isBitSignal = true;
            CDFlib(SELECT_,zVAR_NAME_,"BIT_SIGNAL_SRCID",
                           zVAR_RECNUMBER_,i,
                      GET_,zVAR_DATA_,&BitSignalSrcId,NULL_);
            CDFlib(SELECT_,zVAR_NAME_,"BIT_SIGNAL_MASK",
                           zVAR_RECNUMBER_,i,
                      GET_,zVAR_DATA_,&BitSignalMask,NULL_);
            BitSignalLowBit=0;
            if (BitSignalMask != 0) {
                while ((BitSignalMask & (1 << BitSignalLowBit)) == 0) {
                    BitSignalLowBit++;
                }
            }

            /* Find bit signal source in all zvariables*/
            CDFlib(SELECT_,ATTR_NAME_,"SIGNALID",
                      GET_,CDF_NUMzVARS_,&n,NULL_);
            for (iVar=0 ; iVar<n ; iVar++) {
                CDFlib(SELECT_,zVAR_,iVar,
                            GET_,zVAR_NAME_,TextBuffer,NULL_);
                deblank(TextBuffer);

                /* Find source name of bit signal */
                if ((strncmp(TextBuffer,"_CYCLE",6) != 0) &&
                    (strncmp(TextBuffer,"BIT_SIGNAL_",11) != 0)) {
                    CDFlib(SELECT_,zENTRY_,iVar,
                                GET_,zENTRY_DATA_,&k,NULL_);
                    if (k == BitSignalSrcId) {
                        strcpy(varname,TextBuffer);
                        break;
                    }
                }
            }
            break;
        }
    }

    /* Initialization */
```

```
indices[0]   = 0;
indices[1]   = 0;
counts[0]    = 1;
counts[1]    = 1;
intervals[0] = 1;
intervals[1] = 1;

/* Get name and dimension of channel */
strcpy(TextBuffer,varname);
pt = strchr(TextBuffer,'(');
if (pt != NULL) {
    sscanf(pt,"(%d)",&indices[0]);
    indices[0]--;
    *pt = '\0';
}

/* Read type and number of records for this variable */
status = CDFlib(SELECT_,zVAR_NAME_,TextBuffer,
                  GET_,zVAR_DATATYPE_,&DataType,
                       zVAR_MAXREC_,&np,NULL_);
np++;

/* Variable not found in this file - switch to child file */
if (status != CDF_OK) {
    CDFlib(CLOSE_,CDF_,NULL_);
    if (iFile == 0) {
        printf("zVAR_NAME=%s\n",TextBuffer);
        disperr(status);
    } else {
        iFile--;
        continue;
    }
}

/* Allocate memory for temporary data to read*/
Cycle       = mxCalloc(np,sizeof(int));
VoidDaten   = mxCalloc(np,sizeof(double));
DoubleDaten = (double*)VoidDaten;
FloatDaten  = (float*)VoidDaten;
IntDaten    = (int*)VoidDaten;
UIntDaten   = (unsigned int*)VoidDaten;
ShortDaten  = (short*)VoidDaten;
UShortDaten = (unsigned short*)VoidDaten;
CharDaten   = (char*)VoidDaten;
UCharDaten  = (unsigned char*)VoidDaten;

/* Read data channel */
CDFlib(SELECT_,zVAR_RECNUMBER_,0,
               zVAR_RECCOUNT_,np,
               zVAR_RECINTERVAL_,1,
               zVAR_DIMINDICES_,indices,
               zVAR_DIMCOUNTS_,counts,
               zVAR_DIMINTERVALS_,intervals,
         GET_,zVAR_HYPERDATA_,VoidDaten,NULL_);

/* Read cycle channel */
CDFlib(SELECT_,ATTR_NAME_,"CYCLECHN",
               zENTRY_NAME_,TextBuffer,
          GET_,zENTRY_DATA_,&k,NULL_);
sprintf(TextBuffer,"_CYCLE%d",k);
status = CDFlib(SELECT_,zVAR_NAME_,TextBuffer,
               zVAR_RECNUMBER_,0,
               zVAR_RECCOUNT_,np,
               zVAR_RECINTERVAL_,1,
          GET_,zVAR_HYPERDATA_,Cycle,NULL_);

/* Find start and end cycles for selected ts and te */
if (ts > te) {
    Start = 0;
    End   = np-1;
} else {
    eps = 0.5 * dt;
    for (Start=0 ; Start<np-1 ; Start++) {
        if (ts < UTCTime + (Cycle[Start] - BegCycle)*dt + eps) break;
```

```
    }
    for (End=Start ; End<np-1 ; End++) {
        if (te < UTCTime + (Cycle[End] - BegCycle)*dt + eps) break;
    }
}

/* Allocate memory for interface data */
np   = End - Start + 1;
DATA = mxCreateDoubleMatrix(np,2,mxREAL);
data = mxGetPr(DATA);

/* Compute time axis */
for (i=Start ; i<=End ; i++) {
    *data = UTCTime + (Cycle[i]-BegCycle)*dt;
    data++;
}
/* Evaluate bit signal */
if (isBitSignal) {
    switch (DataType) {
        case CDF_INT4:
        case CDF_UINT4:
            for (i=Start ; i<=End ; i++) {
                IntDaten[i] = (IntDaten[i] & BitSignalMask)
                            >> BitSignalLowBit;
            }
            break;
        case CDF_INT2:
        case CDF_UINT2:
            for (i=Start ; i<=End ; i++) {
                ShortDaten[i] = (ShortDaten[i] & BitSignalMask)
                              >> BitSignalLowBit;
            }
            break;
        case CDF_INT1:
        case CDF_UINT1:
        case CDF_BYTE:
        case CDF_CHAR:
        case CDF_UCHAR:
            for (i=Start ; i<=End ; i++) {
                CharDaten[i] = (CharDaten[i] & BitSignalMask)
                             >> BitSignalLowBit;
            }
            break;
    }
}

/* Typecast data channel to double*/
switch (DataType) {
    case CDF_REAL8:
    case CDF_DOUBLE:
    case CDF_EPOCH:
        for (i=Start ; i<=End ; i++) {
            *data++ = DoubleDaten[i];
        }
        break;
    case CDF_REAL4:
    case CDF_FLOAT:
        for (i=Start ; i<=End ; i++) {
            *data++ = FloatDaten[i];
        }
        break;
    case CDF_INT4:
        *intdata = 1;
        for (i=Start ; i<=End ; i++) {
            *data++ = IntDaten[i];
        }
        break;
    case CDF_UINT4:
        *intdata = 1;
        for (i=Start ; i<=End ; i++) {
            *data++ = UIntDaten[i];
        }
        break;
    case CDF_INT2:
```

```
                    *intdata = 1;
                    for (i=Start ; i<=End ; i++) {
                        *data++ = ShortDaten[i];
                    }
                    break;
                case CDF_UINT2:
                    *intdata = 1;
                    for (i=Start ; i<=End ; i++) {
                        *data++ = UShortDaten[i];
                    }
                    break;
                case CDF_INT1:
                case CDF_BYTE:
                case CDF_CHAR:
                    *intdata = 1;
                    for (i=Start ; i<=End ; i++) {
                        *data++ = CharDaten[i];
                    }
                    break;
                case CDF_UINT1:
                case CDF_UCHAR:
                    *intdata = 1;
                    for (i=Start ; i<=End ; i++) {
                        *data++ = UCharDaten[i];
                    }
                    break;
            }
            mxFree(Cycle);
            mxFree(VoidDaten);
            while (iFile>=0) {
                CDFlib(SELECT_,CDF_,Id[iFile--],
                        CLOSE_,CDF_,NULL_);
            }
        }

/*  Illegal file type */
    } else {
        CDFlib(CLOSE_,CDF_,NULL_);
        mexErrMsgTxt("cdf_hardread: CDF file not standard RCDF");
    }
}
```

A.4 cdf_write.c

```
/*
    cdf_write.c  .MEX file implementation for writing IS3 and RCDF files.

    Author: Achim.Jaekel@dlr.de

        The calling syntax is:
        cdf_write(filename,type,varnames,data,units)

        cdf_write creates a CDF file and writes data into the file.

        INPUT:
          filename  - string containing the name of the CDF file
          type      - string containing the file type ('IS3' or 'RCDF')
          varnames  - string matrix with the channel names
          data      - double data containing the channel data

              units - string matrix with the units (default='-')

        NOTES:
          The first channel must be the time axis.
          For type="IS3", the time channel must have a constant dt.
*/

/* Includes */
#include <math.h>
#include <time.h>
#include <string.h>
#include "mex.h"
#include "cdf_sublib.h"

/* Constants */
#define MAXVAR 10000
#define UNIT_LEN 128

/* Arguments */
#define FILENAME  prhs[0]
#define TYPE      prhs[1]
#define VARNAMES  prhs[2]
#define DATA      prhs[3]
#define UNITS     prhs[4]

void mexFunction(int nlhs,mxArray *plhs[],
                 int nrhs,const mxArray *prhs[])
{
    char   filename[CDF_PATHNAME_LEN+1];         /* CDF file name */
    char   filetype[8];                          /* File type */
    char   varnames[MAXVAR][CDF_VAR_NAME_LEN+1]; /* Varnames */
    double *data;                                /* Channel data */
    char   units[MAXVAR][UNIT_LEN+1];              /* Units */

    int       i,k,BegCycle;
    long      mVar,nVar,mData,nData,mUnit,nUnit,nRun;
    int       hours,minutes,seconds,mseconds;
    long      attrNum,dimSizes[2],dimVarys[2];
    float     run[2],dt;
    double    eps,utctime;
    char      *pt,*buffer;
    CDFstatus status;
    CDFid     id;
    int       *Cycle;
    float     *FloatDaten;
    time_t    actualtime;
    struct tm *kalender;

    /* Argument 1: filename (without extension .cdf) */
    if (nrhs < 1 || (!mxIsChar(FILENAME))) {
        mexErrMsgTxt("cdf_write: Filename argument required");
    }
    if (mxGetN(FILENAME) > CDF_PATHNAME_LEN) {
        mexErrMsgTxt("cdf_write: Filename too long");
    }
```

```
mxGetString(FILENAME,filename,CDF_PATHNAME_LEN+1);
pt=strstr(filename,".cdf");
if (pt != NULL) *pt='\0';

/* Argument 2: type */
if (nrhs < 2 || (!mxIsChar(TYPE))) {
    mexErrMsgTxt("cdf_write: Type argument required");
}
if (mxGetN(TYPE) > 4) {
    mexErrMsgTxt("cdf_write: Type too long");
}
mxGetString(TYPE,filetype,5);
if (strncmp(filetype,"IS3",3) == 0) strcpy(filetype,"IS3 ");
if ((strncmp(filetype,"IS3",3) != 0) && (strcmp(filetype,"RCDF") != 0)) {
    mexErrMsgTxt("cdf_write: CDF file standard unknown");
}

/* Argument 3: varnames */
if (nrhs < 3 || (!mxIsChar(VARNAMES))) {
    mexErrMsgTxt("cdf_write: Varnames argument required");
}
mVar = (long)mxGetM(VARNAMES);
nVar = (long)mxGetN(VARNAMES);
if ((mVar > MAXVAR) || (nVar > CDF_VAR_NAME_LEN)) {
    mexErrMsgTxt("cdf_write: Varnames argument exceeds limits");
}
buffer = mxCalloc(mVar*nVar+1,sizeof(mxChar));
mxGetString(VARNAMES,buffer,mVar*nVar+1);
pt = buffer;
for (k=0 ; k<nVar ; k++) {
    for (i=0 ; i<mVar ; i++) {
        varnames[i][k] = *pt;
        pt++;
    }
}
for (i=0; i<mVar; i++) {
    varnames[i][nVar] = '\0';
    deblank(varnames[i]);
}
mxFree(buffer);

/* Argument 4: data */
if (nrhs < 4 || (!mxIsNumeric(DATA))) {
    mexErrMsgTxt("cdf_write: Data argument required");
}
mData = (long)mxGetM(DATA);
nData = (long)mxGetN(DATA);
if (nData != mVar) {
    mexErrMsgTxt("cdf_write: Number of varnames and data does not correspond");
}
if (mData < 2) {
    mexErrMsgTxt("cdf_write: Data contains less than 2 values per channel");
}
data  = mxGetPr(DATA);

/* Argument 5: units */
if ((nrhs < 5 ) || (!mxIsChar(UNITS))) {
    strcpy(units[0],"s");
    for (i=1 ; i<mVar ; i++) {
        strcpy(units[i],"-");
    }
} else {
    mUnit = (long)mxGetM(UNITS);
    nUnit = (long)mxGetN(UNITS);
    if (nUnit > UNIT_LEN) {
        mexErrMsgTxt("cdf_write: Units argument expands limits");
    }
    if (mUnit != mVar) {
        mexErrMsgTxt("cdf_write: Number of units and varnames does not correspond");
    }
    buffer = mxCalloc(mUnit*nUnit+1,sizeof(mxChar));
    mxGetString(UNITS,buffer,mUnit*nUnit+1);
    pt = buffer;
    for (k=0 ; k<nUnit ; k++) {
```

```
            for (i=0 ; i<mUnit ; i++) {
                units[i][k] = *pt;
                pt++;
            }
        }
        for (i=0; i<mUnit; i++) {
            units[i][nUnit] = '\0';
            deblank(units[i]);
        }
        mxFree(buffer);
    }

    /* Remove CDF, if exist */
    status = CDFlib(OPEN_,CDF_,filename,&id,NULL_);
    if (status == CDF_OK) {
        status = CDFlib(DELETE_,CDF_,NULL_);
        if (status != CDF_OK) disperr(status);
    }

    /* Create CDF */
    status = CDFlib(CREATE_,CDF_,filename,0,0,&id,
                      PUT_,CDF_ENCODING_,HOST_ENCODING,
                           CDF_MAJORITY_,COL_MAJOR,
                           CDF_FORMAT_,SINGLE_FILE,NULL_);
    if (status != CDF_OK) disperr(status);

    /* Create global attribute TITLE */
    CDFlib(CREATE_,ATTR_,"TITLE",GLOBAL_SCOPE,&attrNum,
           SELECT_,gENTRY_,0,
              PUT_,gENTRY_DATA_,CDF_CHAR,6,"DLR-FT",NULL_);

    /* Create global attribute TYPE */
    CDFlib(CREATE_,ATTR_,"TYPE",GLOBAL_SCOPE,&attrNum,
           SELECT_,gENTRY_,0,
              PUT_,gENTRY_DATA_,CDF_CHAR,4,filetype,NULL_);

/*********************/
/* IS3 file standard */
/*********************/
    if (strcmp(filetype,"IS3 ") == 0) {

        /* Create global attribute RECORD_RATE */
        eps = 1.5 * (*(data+1) - *data);
        for (i=1 ; i<mData-1 ; i++) {
            if (*(data+i+1) - *(data+i) > eps) {
                break;
            }
        }
        dt = (float)((*(data+i) - *data) / i);
        CDFlib(CREATE_,ATTR_,"RECORD_RATE",GLOBAL_SCOPE,&attrNum,
               SELECT_,gENTRY_,0,
                  PUT_,gENTRY_DATA_,CDF_FLOAT,1,&dt,NULL_);

        /* Create global attribute RUNS */
        CDFlib(CREATE_,ATTR_,"RUNS",GLOBAL_SCOPE,&attrNum,NULL_);
        nRun = 0;
        run[0] = (float)(*data);
        i = 1;
        while (i<mData-1) {
            if (*(data+i+1) - *(data+i) > eps) {
                run[1] = (float)*(data+i);
                CDFlib(SELECT_,gENTRY_,nRun,
                          PUT_,gENTRY_DATA_,CDF_REAL4,2,run,NULL_);
                run[0] = (float)*(data+i+1);
                nRun++;
            }
            i++;
        }
        run[1] = (float)*(data+mData-1);
        CDFlib(SELECT_,gENTRY_,nRun,
                  PUT_,gENTRY_DATA_,CDF_REAL4,2,run,NULL_);

        /* Create variable attribute UNITS */
        CDFlib(CREATE_,ATTR_,"UNITS",VARIABLE_SCOPE,&attrNum,NULL_);
```

```
        /* Create zvars */
        dimSizes[0] = 1;
        dimVarys[0] = VARY;
        for (i=0 ; i<mVar ; i++) {
            status = CDFlib(CREATE_,zVAR_,varnames[i],CDF_REAL4,1,1,dimSizes,
                                      VARY,dimVarys,&attrNum,NULL_);
            if (status != CDF_OK) {
                CDFlib(CLOSE_,CDF_,NULL_);
                disperr(status);
            }
        }

        /* Save units */
        for (i=0 ; i<mVar ; i++) {
            if (strlen(units[i]) > 0) {
                status = CDFlib(SELECT_,zENTRY_NAME_,varnames[i],
                                PUT_,zENTRY_DATA_,CDF_CHAR,strlen(units[i]),units[i],NULL_);
                if (status != CDF_OK) {
                    CDFlib(CLOSE_,CDF_,NULL_);
                    disperr(status);
                }
            }
        }

        /* Typecast data to float and save it into file*/
        FloatDaten = mxCalloc(mData,sizeof(float));
        for (i=0 ; i<nData ; i++) {
            for (k=0 ; k<mData ; k++) {
                FloatDaten[k] = (float)(*(data++));
            }
            CDFlib(SELECT_,zVAR_,i,
                           zVAR_RECNUMBER_,0,
                           zVAR_RECCOUNT_,mData,
                      PUT_,zVAR_HYPERDATA_,FloatDaten,NULL_);
        }
        mxFree(FloatDaten);

/**********************/
/* RCDF file standard */
/**********************/
    } else if (strcmp(filetype,"RCDF") == 0) {

        /* Create global attribute VERSION */
        CDFlib(CREATE_,ATTR_,"VERSION",GLOBAL_SCOPE,&attrNum,NULL_);

        /* Create global attribute UTCTIME */
        actualtime = time(NULL);
        kalender   = localtime(&actualtime);
        hours      = (int)(*data / 3600);
        minutes    = (int)((*data - 3600 * hours) / 60);
        seconds    = (int)(*data - 3600 * hours - 60 * minutes);
        mseconds   = (int)((*data - 3600 * hours - 60 * minutes - seconds) * 1000);
        utctime = computeEPOCH(kalender->tm_year+1900,
                               kalender->tm_mon+1,
                               kalender->tm_mday,
                               hours,minutes,seconds,mseconds);
        CDFlib(CREATE_,ATTR_,"UTCTIME",GLOBAL_SCOPE,&attrNum,
               SELECT_,gENTRY_,0,
                  PUT_,gENTRY_DATA_,CDF_EPOCH,1,&utctime,NULL_);

        /* Create global attribute DMCCYCLE */
        dt = 0.001f;
        CDFlib(CREATE_,ATTR_,"DMCCYCLE",GLOBAL_SCOPE,&attrNum,
               SELECT_,gENTRY_,0,
                  PUT_,gENTRY_DATA_,CDF_FLOAT,1,&dt,NULL_);

        /* Create global attribute BEGCYCLE */
        BegCycle=1;
        CDFlib(CREATE_,ATTR_,"BEGCYCLE",GLOBAL_SCOPE,&attrNum,
               SELECT_,gENTRY_,0,
                  PUT_,gENTRY_DATA_,CDF_INT4,1,&BegCycle,NULL_);

        /* Create global attribute COMMENT */
```

```
CDFlib(CREATE_,ATTR_,"COMMENT",GLOBAL_SCOPE,&attrNum,
       SELECT_,gENTRY_,0,
          PUT_,gENTRY_DATA_,CDF_CHAR,36,
          "This file was generated by cdf_write",NULL_);

/* Create global attribute FILENAME */
pt = strrchr(filename,filesep());
if (pt == NULL) {
    pt = filename;
} else {
    pt++;
}
CDFlib(CREATE_,ATTR_,"FILENAME",GLOBAL_SCOPE,&attrNum,
       SELECT_,gENTRY_,0,
          PUT_,gENTRY_DATA_,CDF_CHAR,strlen(pt),pt,NULL_);

/* Create variable attribute SIGNALID */
CDFlib(CREATE_,ATTR_,"SIGNALID",VARIABLE_SCOPE,&attrNum,NULL_);

/* Create variable attribute SIGUNIT */
CDFlib(CREATE_,ATTR_,"SIGUNIT",VARIABLE_SCOPE,&attrNum,NULL_);

/* Create variable attribute CYCLECHN */
CDFlib(CREATE_,ATTR_,"CYCLECHN",VARIABLE_SCOPE,&attrNum,NULL_);

/* Create zVARs */
dimSizes[0] = 1;
dimVarys[0] = VARY;
CDFlib(CREATE_,zVAR_,"BIT_SIGNAL_ID",CDF_INT4,1,0,dimSizes,
               VARY,dimVarys,&attrNum,NULL_);
CDFlib(CREATE_,zVAR_,"BIT_SIGNAL_NAME",CDF_CHAR,64,0,dimSizes,
               VARY,dimVarys,&attrNum,NULL_);
CDFlib(CREATE_,zVAR_,"BIT_SIGNAL_SRCID",CDF_INT4,1,0,dimSizes,
               VARY,dimVarys,&attrNum,NULL_);
CDFlib(CREATE_,zVAR_,"BIT_SIGNAL_MASK",CDF_UINT4,1,0,dimSizes,
               VARY,dimVarys,&attrNum,NULL_);
CDFlib(CREATE_,zVAR_,"_CYCLE1",CDF_INT4,1,0,dimSizes,
               VARY,dimVarys,&attrNum,NULL_);
for (i=1 ; i<mVar ; i++) {
    status = CDFlib(CREATE_,zVAR_,varnames[i],CDF_REAL8,1,0,dimSizes,
                            VARY,dimVarys,&attrNum,NULL_);
    if (status != CDF_OK) {
        CDFlib(CLOSE_,CDF_,NULL_);
        disperr(status);
    }
}

/* Save signal IDs */
CDFlib(SELECT_,ATTR_NAME_,"SIGNALID",NULL_);
k = 0;
for (i=1 ; i<mVar ; i++) {
    CDFlib(SELECT_,zENTRY_NAME_,varnames[i],
              PUT_,zENTRY_DATA_,CDF_INT4,1,&k,NULL_);
}

/* Save units */
CDFlib(SELECT_,ATTR_NAME_,"SIGUNIT",NULL_);
for (i=1 ; i<mVar ; i++) {
    if (strlen(units[i]) > 0) {
        CDFlib(SELECT_,zENTRY_NAME_,varnames[i],
                  PUT_,zENTRY_DATA_,CDF_CHAR,strlen(units[i]),units[i],NULL_);
    }
}

/* Save cycle channel numbers */
CDFlib(SELECT_,ATTR_NAME_,"CYCLECHN",NULL_);
k = 1;
for (i=1 ; i<mVar ; i++) {
    CDFlib(SELECT_,zENTRY_NAME_,varnames[i],
              PUT_,zENTRY_DATA_,CDF_INT4,1,&k,NULL_);
}

/* Typecast cycle data to int and save*/
Cycle = mxCalloc(mData,sizeof(int));
```

```
        for (i=0 ; i<mData ; i++) {
            Cycle[i] = (int)((*(data+i) - *data) / dt + 0.5) + BegCycle;
        }
        CDFlib(SELECT_,zVAR_NAME_,"_CYCLE1",
                       zVAR_RECNUMBER_,0,
                       zVAR_RECCOUNT_,mData,
                  PUT_,zVAR_HYPERDATA_,Cycle,NULL_);
        mxFree(Cycle);

        /* Save channel data*/
        for (i=1 ; i<mVar ; i++) {
             CDFlib(SELECT_,zVAR_NAME_,varnames[i],
                            zVAR_RECNUMBER_,0,
                            zVAR_RECCOUNT_,mData,
                       PUT_,zVAR_HYPERDATA_,data+i*mData,NULL_);
        }
    }

    /* Close CDF */
    status = CDFlib(CLOSE_,CDF_,NULL_);
    if (status != CDF_OK) disperr(status);
}
```

A.5 cdf_sublib.c

```
/*-------------------------------------------------------------------*/
/* Subroutines used of cdf_head, cdf_read, cdf_hardread, cdf_write */
/* Author: Achim.Jaekel@dlr.de                                       */
/*                                                                   */
/* void handle(CDFstatus status) : Output of status text             */
/* char* deblank(char* text)     : Cut trailing blanks of string     */
/* char  filesep()               : Get the systems file separator    */
/*-------------------------------------------------------------------*/
#include <string.h>
#include "mex.h"
#include "cdf.h"

/* CDF status error output */
void disperr(CDFstatus status)
{
    char statustext[CDF_STATUSTEXT_LEN+1];

    CDFerror(status,statustext);
    mexErrMsgTxt(statustext);
}

/* Cut trailing blanks */
char* deblank(char* text)
{
    while ((strlen(text) > 0) && (text[strlen(text)-1] == ' ')) {
        text[strlen(text)-1] = '\0';
    }

    return (text);
}

/* Get the systems file separator */
char filesep()
{
    char    TextBuffer[2];
    mxArray *lhs[1];

    mexCallMATLAB(1,lhs,0,NULL,"filesep");
    mxGetString(lhs[0],TextBuffer,2);
    mxDestroyArray(lhs[0]);

    return (TextBuffer[0]);
}
```

A.6 cdf_sublib.h

```
/* Header file for cdf functions for fitlabGui */
#include "cdf.h"

/* Forward declarations */
void disperr(CDFstatus);
char* deblank(char*);
char filesep();
```

A.7 make_cdf.m

```
% make_cdf
% Compilieren der CDF-Schnittstellen fuer fitlab
%
% Autor: Achim Jäkel
% Datum: 07.09.2015

% CDF Includeverzeichnis und Bibliotheksverzeichnis festlegen
disp(['Rechnerarchitektur: ',computer]);
disp('==========================');
if strcmp(computer,'PCWIN')         % Windows MATLAB 32 Bit
    CDFINC = '..\CDF\w32\';
    CDFLIB = '..\CDF\w32\';
elseif strcmp(computer,'PCWIN64')   % Windows MATLAB 64 Bit
    CDFINC = '..\CDF\w64\';
    CDFLIB = '..\CDF\w64\';
elseif strcmp(computer,'GLNXA64')   % Linux MATLAB 64 Bit
    CDFINC = '/usr/local/include';
    CDFLIB = '/usr/local/lib64';
elseif strcmp(computer,'GLNX86')    % Linux MATLAB 32 Bit
    CDFINC = '/usr/local/include';
    CDFLIB = '/usr/local/lib';
else
    disp('ERROR: Unbekannte Rechnerarchitektur');
    return;
end

% Compilieren
eval(['mex cdf_head.c     cdf_sublib.c -lcdf -v -I',CDFINC,' -L',CDFLIB]);
eval(['mex cdf_read.c     cdf_sublib.c -lcdf -v -I',CDFINC,' -L',CDFLIB]);
eval(['mex cdf_hardread.c cdf_sublib.c -lcdf -v -I',CDFINC,' -L',CDFLIB]);
eval(['mex cdf_write.c    cdf_sublib.c -lcdf -v -I',CDFINC,' -L',CDFLIB]);
```