CFD通用符号系统

高级数据格式（ADF）用户指南

文档版本：V1.1.1

翻译： 毛仲军 刘云楚
审校： 刘云楚

注：由于时间仓促，加上水平有限，错误在所难免，恳请批评指正。

目录

1 引言

高级数据格式（ADF）是一种基本的数据库管理和 I/O 子程序的集合，它实现相对简单的层次数据库。ADF 用 ANSI C 语言编制以加强软件的可移植性和方便数据库文件在这种平台间移植。当然，它会有一个格式接口，文件具有自我描述和可扩展性。

程序允许用户为他们的数据建立一个树性的结构。（如图 1）这种结构与 UNIX 和 DOS 操作系统的目录结构非常相似。ADF 允许在相同的文件之间或不同文件之间建立结点链接。这种功能有点像 UNIX 下的软连接（soft links）。前者与 ADF 主要的不同点在于 ADF 不仅包含了子目录（next low-level nodes）的信息还包含了带数据信息的子结点。

安装包里包含了 ADF 格式、Fortran 语言接口、示例文件和一个简单文件浏览器。

1.1 项目的沿革

ADF 是 CFD 通用符号系统（CGNS）项目的一部分。CGNS 项目的目的是为了基于 CFD 的 N—S 方程建立数据模型，为数据模型发展一套标准的数据结构接口，发展可以利用这些数据结构为现有和将来的 CFD 分析工具服务的软件系统。CNGS 系统由 CFD 领域内的约定和习惯组成，软件兼容这些约定来存储和修补 CFD 数据。坚持使用约定俗成是为了方便在在各种站点、软件代码（如各种解算器、网格生成器）、计算平台间交换 CFD 数据。

一旦数据模型（标准数据结构，SIDS）被定义出来，就启动一个能够完整再生磁盘信息的软件编制工程。ADF 就是这样一工程的成果。由于 ADF 是特地为 CFD 程序发展的，因此 ADF 是通用的和没有内建 CFD 知识的；因此，ADF 对于存储任何类型数据和传递给分级定义具有很强的适用性。

1.2 其他软件接口与 ADF

在这个项目中，其他两种软件包作为工程的一部分进行了研究。第一种数据库接口层次数据格式(HDF)是在 Illinois 州立大学 National Center for Supercomputing Applications 研究的。该数据库拥有庞大的用户和支持者，已经问世六年，采用 C 语言和 Fortran 语言编写，拥有工具化和图形化的子程序。任何层级都使用约定俗成的命名来建立。既然 CNGS 数据模型指向一个自然的分层结构，那么，通过设计发展一个这种模式的数据库软件是相当适当的。ADF 设计的想法参考附录 E。

第二种数据库接口是一般文件格式（Common File Format，CFF）是麦道公司发展的。CFF 是第二代数据库管理软件，可为 CFD 数据提供统一的文件结构。发展 CFF 目的就是为让用户从无数的计算机类型中解脱出来，可以让用户或编程者不用对来自其它机器的文件进行复杂的转化就可方便的使用。用 Fortran 语言编制的 CFF，可以通过 C 语言加强其可移植性和可扩展性。ADF 格式大量吸收了麦道集团 CFD 团队员工的经验。

1.3 本指南的结构介绍

本指南主要篇幅用在了 ADF 数据库的分层结构上。此外，还着重介绍了作为构成分层结构基本要素的结点（node）概念。其余的部分涉及范围也是相当广泛的，包括了大量的术语和约定以及 ADF 的各种版本信息。

2 ADF 软件库

ADF 软件库是围绕“结点”概念建立的一种分层数据库系统。每一个结点包含了它自身和它的父结点的信息以及可能的数据（如数组、矢量、字符串等等）。每一个结点从顺序上来说都可以和任意数量的子结点相连接，每个子结点本身也是结点。在本系统中，ADF 结点含有用户可存取的包括如 ID、名称、类型、相关数据量以及指向子结点指针的信息。基本的结点包括以下信息：

- 一个独一无二的 ID 码（本质上说就是文件的地址指针）
- 一个描述结点和数据的名称（特定的字符串）
- 一个附加字段，用来描述结点和它的数据的标签（特定字符串），有点类似名字（name），但不完全是。
- 描述类型和数据量的信息
- 数据
- 子结点的 ID

在 ADF 格式中，对子结点的数量并没有进行限制。这种结构允许建立如图 1 中的那种分层结构。正如图所示，可以从原始文件（ADF_File_One）参考到第二个文件（ADF_File_One）的结点，就是所谓的连接（linking）概念。

一个结点知道它和它自结点的信息，但它不知道任何有关它父结点的信息，这意味着它可以通过询问的方式访问到向下的树结构，却无法向上访问。如果想向上访问，用户就必须自己维护一个路径信息。

所有的 ADF 都是从一个叫根结点（ADF MotherNode）的结点开始的。根结点在 ADF 文件生成时自动生成。根结点在 ADF 文件中是唯一的。

2.1 ADF 结点属性

如图 1 所示有一个块的结点来构造分层结构，每一个结点都有零到很多结点与之相连，下面列出了在 ADF 分层数据库结构中用户可以访问到的结点属性。

- Child Table　　结点的子结点的文件指针和名称的列表
- Data　　结点相关的数据
- Data Type　　一个32字节的字符字段，指定数据的类型 （如 Real ，Integer ，Complex)。支持的数据结构见表1。
- Dimensions　　整数矢量，含有每一个维度的元素个数。比如，Fortran中数组A（10，20），Dimension就含有两个值（10，20）

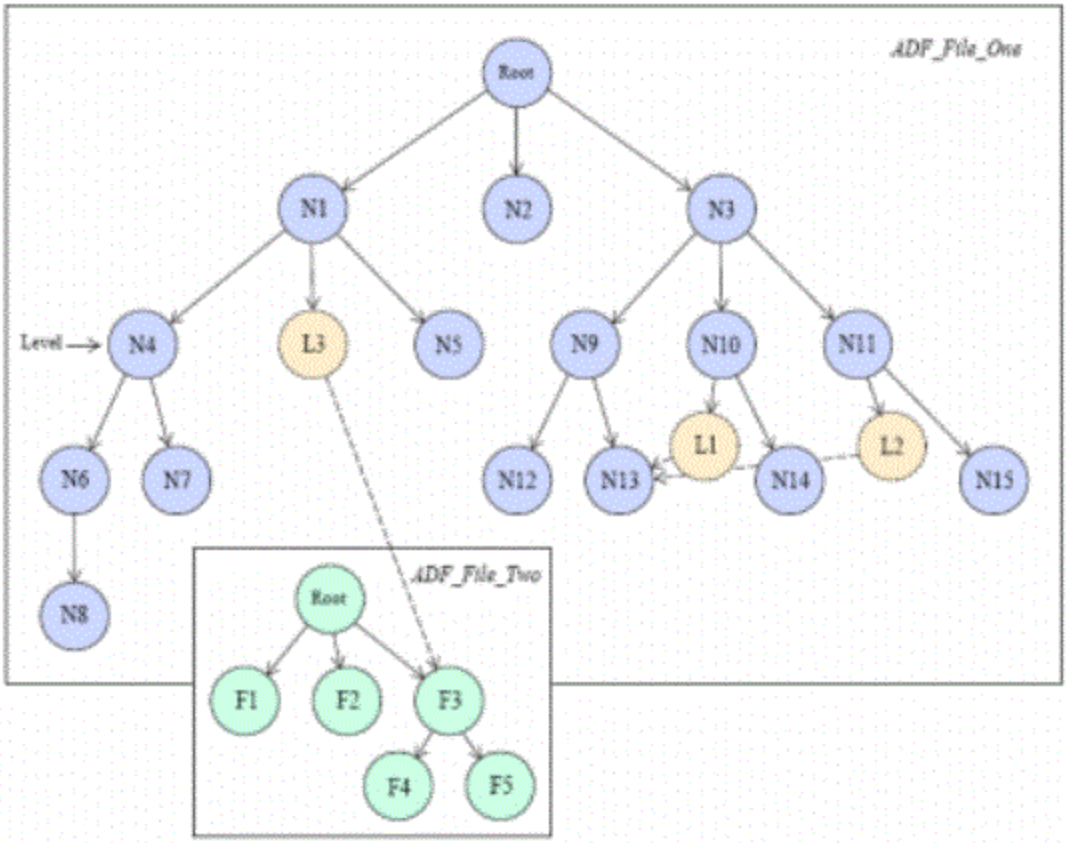

Figure 1: Example Database Hierarchy of Nodes

●ID　文件中存取结点的一个的独一无二的身份 ID 号。这个字段有在文件中定位结点的充分信息。对于一个给定的结点，ID 会在结点所在的文件被程序和用户调用结点信息时生成。ID 只有在程序打开文件和文件被打开时才具有合法性，如果文件被关闭和重新打开，其 ID 会不同。在不同的程序中，同一个结点的 ID 是不同的。ID 实际上是不写入文件的。

●Label　32 字节的字段。Labels 的规则与名称(name)是一样的，不同用于名称，Label 不一定是唯一的。Label 引入成为 “数据类型”的说明与 C 语言的“Typedef”概念相当。使用 Label 字段可以让程序了解结点和它的子结点的更多信息，可以在侦测到结点类型后调用特定的子程序并做出特定的反应。

Table 1: Data Types

Data Type	Notation
No Data	MT (i.e., eMpTy)
Integer 32	I4
Integer 64	I8
Unsigned Integer 32	U4
Unsigned Integer 64	U8
Real 32	R4
Real 64	R8
Complex 64	X4
Complex 128	X8
Character (unsigned byte)	C1
Byte (unsigned byte)	B1
Link	LK

●Name 32 字节的字符字段。与父结点相连的子结点的名称必须是唯一的。比如，图 1 中所有与结点 N3 相连的结点必须拥有唯一的名称。当需要建立一个新的结点的时候，ADF 会检查请求的是否与特定的父结点所有的子结点名称冲突，如果有冲突 ADF 就返回一个错误。

合法的名称字符是 A-Z，0-9,特定字符（32-126 的 ASCII 值，除了 ASCII 值为 47 的“/”)。名称是 32 字节表格,而且区分大小写。抬头的空格(Leading blank）废弃不用，（trailing blank）忽略，然而，内部的空格（internal blank）应该是标记明显的。

需要注意的是：使用 C 语言时候，传给 ADF 的名称需要在名称后面附上空字符“/0”，然而，C 语言程序中需要安排 33 字节的空间，因为对所有名称（Name）来说都要加上空字符“/0”。

在 Fortran 语言中为 Name 保留了 32 字节的空间，Fortran 语言接口考虑到了增减空字符的问题。

●Name of Subnodes 一个结点的子结点名称的列表。

●Number of Dimensions 数据的维度。ADF 视所有的数据为一个数组（Array），能处理 0 维（没有数据的情况）到 12 维的数据。如果数据为空则用“0”表示。一个标量则表示为维度为 1 的矢量，其长度为 1。

●Number of Subnodes 给定的任一结点的子结点数量。每一个结点都拥有 0 或多个字结点。

●pointer 从编程语言的角度看，就是一个地址。指针就是指从数据这部分

到数据的另一部分的步长。

2.1.1 数据类型的定义

- Structure　在ADF中定义“Structure”与C语言中定义“Struct”很相似。ADF会把structure实例当成单个的数据实列。结构由一字符串定义，如“I4，I4,R8,R4”。

注意：

I. Structures 只能用 32 字节进行描述；

II. 这种构造的移植性很差，不鼓励使用。

- Character　字符类型的数据按照ASCII类型字符信息一致设计的。很多不同系统架构的系统采用不同数据翻译系统。字节应该按纯字节或比特数据使用。
- Precision　在个别的32位架构的系统中，R4和R8是单双 精度。在Cray机上，单精度是R8，双精度可能是R8，也可能是R16，取决于编译器的设定。ADF追踪字节的长度来保证精度。
- Link　link在Table 1中表示为“LK”，定义了结点和子结点之间的链接，在ADF中link用在层级和文件不同的部分连接结点。ADF Link只会在连接的结点的信息被请求时才进行解析。

2.2 获得的软件和文档

ADF 软件是作为 CGNS 库的一部分发布的，可以从 SourceForge 可以得到，网站为：http://sourceforge.net/projects/cgns.

本手册以及 CGNS 的其它文档的 Html 和 Pdf 格式文档在可以在网站得到，http://www.grc.nasa.gov/www/cgns/

ADF 术语汇编

ADF　　Advanced Data Format（高级数据格式）

Child　　父结点的子结点，子结点没有其父结点的任何信息。

Database　ADF 结点的层级结构，通过运用 Link 可以跨越不同的文件。

File　　ADF 文件，拥有一个根结点及其下级结构。

ID　　一个文件的结点的独一无二的身份 ID 号。这个 8 字节的字段内含有在文件中定位结点的充分信息。对于一个给定的结点，ID 会在结点所在的文件被程序和用户调用结点信息时生成。ID 只有在程序打开文件和文件被打开时才具有合法性，如果文件被关闭和重新打开，其 ID 会不同。在不同的程序中，同一个结点的 ID 是不同的。ID 实际上是不写入文件的。

Link-node　一种特殊类型的结点。Links 可以通过 ADF_link 子程序生成。这种结点的类型是 LK，它的数据是一维数组，含有文件名称，相连的结点，以及从根结点到目标结点的全路径。

Link 提供了一种指向不同的分层结构的结点的机制，被 Link 指向的结点可以与 link 同在一文件之中，也可以不在同一文件之中。ADF 中的 link 与 UNIX 中软连接（soft link）相似，但 UNIX 并不保证参考结点存在。ADF Link 只会在连接的结点的信息被请求时才进行解析。Link-node 的 ID 在 ADF 中被调用，即相当于 Link-node 指向的结点被 ADF 调用。需要注意的是，link-node 自身并没有子结点。在图！中，L3 的下级为 F4 和 F5。如果一个子结点添加给 L3，其实是加给了 F3。有特别的子程序用来建立 Link-node 和解析连接信息的详情。

Node　　用来建立 ADF 数据库的基本单位。

Node name　32 字节的字段。与父结点相连的子结点的名称必须是唯一的。合法的名称字符是 A-Z，0-9,特定字符（32-126 的 ASCII 值，除了 ASCII 值为 47 的“/”）。名称是 32 字节表格，而且区分大小写。抬头的空格（Leading blank）废弃不用，（trailing blank）忽略，然而，内部的空格（internal blank）应该是标记明显的。

Parent　　拥有与其直接相连子结点的结点。

Pathname　在数据库中，一个结点的定位可以通过其父结点的 ID 所含的信息进行定位，也可以通过“Pathname”来定位，pathname 的语法与 UNIX 环境下的 pathname 有点类似。一个 pathname 是以文件的根结点“/”开头。如果没有“/”开头，默认是从其父结点的名称开始。尽管结点名有 32 字节的空间限制，但 Pathname 没有长度限制。例如，图 1，同样的指向 N8：

- Node-ID for `N6` and name = "`N8`"
- Node-ID for `N4` and name = "`N6/N8`"
- Node-ID for `N1` and name = "`N4/N6/N8`"
- Node-ID for the `Root_Node` and name = "`/N1/N4/N6/N8`"

B　ADF 的版本历史

这部分包括了自 1995 年 ADF 库的第一个版本发布以来的所有版本清单。我们提出了各版本的升级大纲。将来，更新的版本将可以从互联网上得到。

（以下略）

C ADF 文件系统结构

下列平台架构，已经经过原型的测试和运行验证。

Table 2: Platform Architectures

Release	Machine	OS Version	Native Format
A01	Cray	Unicos 8.0	Unicos 8.0
A01	SGI/IRIS	4.0.5	IEEE Big Endian[1]
B01	HP	9.05	IEEE Big Endian
B01	SGI/IRIS	5.03	IEEE Little Endian
C00	Intel Paragon	—	IEEE Little Endian
C00	Dec Alpha	—	IEEE Little Endian
C00	SGI/IRIS	6.2	IEEE Big Endian
C00	Cray T90	Unicos 9.02	Cray Format

D　ADF 文件版本控制编号

ADF 的文件版本控制描述如下，格式表示为：

AXXxxx

其中

A　主要的修订号。主要的内部结构变化。这个号是不经常变化的，只在明显的策略变化时，改变了向后兼容性。

变化范围:A-Z,a-z

XX　小幅调整版本号。新功能、小幅变化、bug 修复，向后兼容非向前兼容。

变化范围：00-FF

xxx　incremental number

变化范围：000-fff

向前兼容（Forward compatible）　老版本库可以读写更新的版本库创建的文件

向后兼容（Backward compatible）　新版本库可以读写老版本库创建的文件。

E　ADF 设计思想

此部分介绍设计 ADF 软件库的想法。

E. 1　ADF 头文件信息

每一个 ADF 文件都有一个用来存放文件信息的区块。下面这些信息是用户可以访问的 ：

- 创建该数据库的ADF版本信息
- 文件的生成时间和日期
- 文件的修改时间和日期
- 在数据库中的数据格式(IEEE big endian, IEEE little endian, etc.)

E. 2　ADF 文件优化

为了优化 ADF 的性能，提高加强文件的运行能力，下面这些技术加入了 ADF 中：

- 采用了基于区块（block）的、非缓冲（raw mode）I/O,区块大小为4096字节。
- 在区块大小限制内组合中大型数据。
- 如果数据的大小等于或大于2048字节
- 一紧密排列数据到下一区块（next block）
- 一多出的部分放到自由列表（garbage）中
- 尽可能避免小块和不大的数据块跨越区块（block）边界
- 紧密排列到下一区块，多出的部分放到自由列表　　（garbage）中

允许通过连接大块的数据块扩展数据空间。也可以增大最后一个维度(last dimension）的大小来扩展数据。然而，对内部维度（internal dimension ）的改变将破坏已有数据。

子结点的指针列表也将包含子结点的名称。

E. 3　ADF 文件的可移植性

考虑到地址代码的可移植性和未来的需要，加入了下列设计：

- 采用大于32比特的文件指针来使文件的大小可以达到4GB.(C程序lseek可能无法处理，但ADF文件允许)
- 在数据区块内使用48比特的指针。32比特指针指向4096字节的数据块，一个16比特的指针指向块内的一个位置。
- ID指针将是64比特的ID.用户可以采用双精度类型（double real*8)。其中32比特用于区块数，12比特用于区块位移（offset）,20bit用于文件ID。
- 对整数信息进行编码，不同于数据，是用ASCII码，十六进制符号。

E. 4　ADF 文件错误检查（eror checking）

错误检查集成在了ADF文件中。ADF文件中的每一item都有围绕它的特定的标签提供基于文件的损坏检查。对于大小不定的items，相关的边界标签会有基于文件的大小信息。信息将会按顺序写入磁盘中，但损坏的文件将不会写入。例如，当需要增加一个子结点时，在父结点的子结点列表更新前，子结点的信息将被写入。将提供一个装载数据到磁盘的以ADF为核心的子程序。

E.5 ADF 源代码的考虑

ADF的源代码版本信息将与UNIX系统中的“WHAT”字串和RCS的版本信息在源代码和目标代码和并兼容。源代码采用ANSI C编写，采用POSIX定义的系统调用方式。

E.6 ADF 结点头（header）信息

ADF结点头包含一下信息：

- 结点边界标签（boundary tags）
- Name (32字符)
- Label (32字符)

—子结点的数量(num_sub_nodes)类型为整数（integer）；

—子结点的地址采用变量表示。

- sub_nodes, 是文件指针的列表，结点的所有子结点的名称。注意子结点的名称信息对运行性能是多余的。
- 数据类型定义为data_type(32 characters)。
- 数据的维度为num_dimensions（ASCII,HEX编码的）
- 维度的值列表在一个整数12的数组中，dimension_values，HEX编码的。
- 数据簇（data chunks）的数量在num_data_chunks中可以找到（ASCII,HEX编码的）。
- Data address

—如果num_data_chunks = 1,文件指针指向数据；

—如果num_data_chunks > 1,文件指针指向名为num_data_chunks的文件指针表和相关文件的大小值。

- 结点边界的结束标签（ending node boundary tag）。

E.7 Fortran 语言的字符数组的可移植性考量

Fortran的字符数组不同于其他的数组，因为其含有字符长度信息。概括地说，它们是一种复合类型：数组和整数。ANSI标准中并未特别的设置处理这种类型的机制，所以留给了软件提供商。可以想象，软件提供商肯定会设计出不同的处理方式，特别是，参数(argument)的创建和使用，这种情形在Fortran调用有其他语言编写的写入功能显得特别的麻烦。

为了保持接口的简单和Fortran与C语言的I/O调用的一致（as opposed to having separate data I/O functions for character data),ADF要求满足以下规则（这些是Cary T-90mode的用户需要的）：

- 不要把任何字符数组当成实际的数据变量给任何ADF读写函数，除非结点是用C1类

型定义的；

- 如果结点是用C1类数据类型定义的，那么只能把字符数组按照实际的数据变量传递给ADF读写函数。

Cary T-90的用户需要特别注意：上面的规则必须要遵循。另外，给定的结点要是可用的和数据类型特别定义的。出错处理是不可能的，ADF会不管任何情况而退出。

E.8 Integer 64 数据类型的移植性考量

为了移植性的考虑，强烈建议 I8 数据类型的使用严格限制在 64-bit 的环境中。

E.9 复合型数据类型的可移植性考量

基于传递性的考虑，不推使用荐复合型的数据类型。

当使用使用复合型类型数据(如C中的structures)时，了解数据的联合声明(alignment issues）是相当重要的。如果不注意，在存储器中的 structure 的实际大小可能会超过其下的各量的总和。总的大小取决于特定数据类型排列顺序和文字边界。这与操作系统和编译器决定的而非 ADF 来处理。为了提供最大的移植性（至少达到 64-bit 环境），建议：

- 8-byte数据类型（I8,R8）必须与8byte的边界(boundaries)相一致；
- 小于字大小（word size）的数据类型需补足与字大小相同。

因此，当遵循 8-byte 数据类型时，4-byte 的数据类型（I4(1 足(I4(2)。假如一个字的长度为 4 字节，那么所有的 C1-data-type 元素的维度应该是 4 字节的倍数（如，C1(4),C1(8)）。为了达到更好的一致性和谨慎，都以 8 字节为倍数是比较好的“C1[8],I4[2],C1[16],R4[6],R8[5],I8[1]”。

对于给定的架构和编译器，严格考虑上述限制，复合型的数据类型的使用是可以的。用户将个人的结构（structure）组件写到独立的结点，就能得到更好的移植性。

F　ADF 术语和使用

所有的输入字符串都必须以空字符(null) 结束。

所有返回的字符串的尾部空白是去掉的，且以空字符结尾。Names、Label 和 Data-type 变量的长度应该为 33 characters 长。ADF.h 定义了变量的数量。一个实例如下：

charname[ADF_NAME_LENGTH+1];

Fortran　字符串的长度由继承的长度决定。返回的字符串要用空白填充到特定的长度。所有返回的 names 要经判断进行正确的填充。是没有空字符的（null character)。例如：

PARAMETERADF_NAME_LENGTH=32

CHARACTER*(ADF_NAME_LENGTH)NAME

ID　一个文件的结点的独一无二的身份 ID 号。这个区域内含有在文件中定位结点的充分信息。对于一个给定的结点，ID 会在结点所在的文件被程序和用户调用结点信息时生成。ID 只有在程序打开文件和文件被打开时才具有合法性，如果文件被关闭和重新打开，其 ID 会不同。在不同的程序中，同一个结点的 ID 是不同的。ID 实际上是不写入文件的。

存放 ID 变量应该是 8-byte 的实数(real number)。ID 其实是一个 64-bit 的系统生成文件索引和在磁盘上区块位置偏移的结合体。一般来说，拥护不需要知道内部的信息解码过程。

error_return　ADF 程序的错误码，意义如下：

-1　没有错误；

n(>0)　ADF 错误（error)。子程序 ADF_Error_Message 用来获得错误的文本描述；

ADF 应该永远不会返回 0 值。

Indexing　所有的索引（indexing）和 Fortran 中的相似，N 取值

1 到 N,N 为索引的长度或数组维度。索引最先被使用的是 ADF_read_data, ADF_write_data, ADF_children_names。

用户需要知道 Fortran 和 C 语言之间的数组索引的不同点。子程序 ADF_read_data, ADF_write_data 仅仅是将指针指向数据的起始点，计算有多少数据需要读写，处理被请求的数据。因而，这些子程序非常有效的对内存作一份备份到硬盘。对于这个约定，C 程序可能是采用 C 标准的约定的数组索引，用 ADF_Write_All_Data 存储数组到磁盘。Fortran 程序会使用 ADF_Read_All_Data 来读取数据集。除非用户知道数据的 structure，数组应该转化到用户期望的样子。

设定的数组结构约定的意味是相当微妙的和精细的。为了索引数据，子程序 ADF_read_data 和 ADF_write_data 采用 Fortran 数组结构（Structure)。

除非用户知道这点，否则用户可能会向磁盘写入数组，而后又改变数组的数据比例和不改变正确数量。只要用户知道 ADF 上的数据结构图，就不会有任何问题。

G ADF 错误信息 (error messages)

Table 3: ADF Error Messages

Number	Error Message
-1	No Error
1	Integer number is less than a given minimum value
2	Integer value is greater than given maximum value
3	String length of zero of blank string detected
4	String length longer than maximum allowable length
5	String length is not an ASCII-Hex string
6	Too many ADF files opened
7	ADF file status was not recognized
8	ADF file open error
9	ADF file not currently opened
10	ADF file index out of legal range
11	Block/offset out of legal range
12	A string pointer is null
13	FSEEK error
14	FWRITE error
15	FREAD error
16	Internal error: Memory boundary tag bad
17	Internal error: Disk boundary tag bad
18	File Open Error: NEW - File already exists[2]
19	ADF file format was not recognized
20	Attempt to free the RootNode disk information
21	Attempt to free the FreeChunkTable disk information
22	File Open Error: OLD - File does not exist[3]
23	Entered area of unimplemented code
24	Subnode entries are bad
25	Memory allocation failed
26	Duplicate child name under a parent node
27	Node has no dimensions
28	Node's number of dimensions is not in legal range
29	Specified child is not a child of the specified parent
30	Data-Type is too long
31	Invalid Data-Type
32	A pointer is null

Continued on next page

[2]The user is trying to create a new file and give it a name. The system has responded that the name has already been used.

[3]The user wants to open an existing file that supposedly has the given name. The system has responded that no file by that name exists.

Table 3: **ADF Error Messages** (*Continued*)

Number	Error Message
33	Node had no data associated with it
34	Error zeroing out of memory
35	Requested data exceeds actual data available
36	Bad end value
37	Bad stride values
38	Minimum value is greater than maximum value
39	The format of this machine does not match a known signature[4]
40	Cannot convert to or from an unknown native format
41	The two conversion formats are equal; no conversion done
42	The data format is not supported on a particular machine
43	File close error
44	Numeric overflow/underflow in data conversion
45	Bad start value
46	A value of zero is not allowable
47	Bad dimension value
48	Error state must be either a 0 (zero) or a 1 (one)
49	Dimensional specifications for disk and memory are unequal
50	Too many link levels are used; may be caused by a recursive link
51	The node is not a link. It was expected to be a link.
52	The linked-to node does not exist
53	The ADF file of a linked node is not accessible
54	A node ID of 0.0 is not valid
55	Incomplete data when reading multiple data blocks
56	Node name contains invalid characters
57	ADF file version incompatible with this library version
58	Nodes are not from the same file
59	Priority stack error
60	Machine format and file format are incomplete
61	Flush error

H. 默认常值、默认大小、数组和维度的限量

下表中的默认常值、默认大小、数组维度限量在 ADF.h 中定义：

Table 4: Default Values and Sizes

Attribute	Limit, Size, or Value
Data type length	32-byte character field
Date length	32-byte character field
File name length	1024-byte character field
Format length	20-byte character field
Label length	32-byte character field
Maximum link depth	100
Maximum dimension	12
Maximum length of error string	80 characters
Maximum link data size	4096
Name length	32-byte character field
Length of status	32-byte character field
Version length	32-byte character field
Maximum children	None
Pointer size	48-bit pointer is used for a block size of 4096 32-bit pointer is used for blocks less than 4096 16-bit pointer is used for blocks within blocks

ADF 子程序库

本附录列出了 ADF2.0 版本的子程序集。对其在 Fortran 和 C 语言中的使用进行了详细地介绍。

I.1 主要的函数集

Database-Level Routines

C	Fortran	Description	Page
ADF_Database_Close	ADFDCLO	Close an opened database	31
ADF_Database_Delete	ADFDDEL	Delete an existing database[5]	32
ADF_Database_Get_Format	ADFDGF	Get the data format in existing database	33
ADF_Database_Open	ADFDOPN	Open a database	29
ADF_Database_Set_Format	ADFDSF	Set the data format in an existing database	34

Data Structure and Management Routines

C	Fortran	Description	Page
ADF_Create	ADFCRE	Create a node	35
ADF_Delete	ADFDEL	Delete a node	39
ADF_Children_Names	ADFCNAM	Get the child names of a node	43
ADF_Number_of_Children	ADFNCLD	Get the number of children of a node	47
ADF_Get_Node_ID	ADFGNID	Get a unique identifier of a node	48
ADF_Get_Name	ADFGNAM	Get the name of a node	51
ADF_Put_Name	ADFPNAM	Put (change) the name of a node	52
ADF_Move_Child	ADFMOVE	Change a parent (move a child)	55
ADF_Link	ADFLINK	Create a link	59
ADF_Is_Link	ADFISLK	Test if a node is a link	66
ADF_Get_Link_Path	ADFGLKP	Get the path information from a link	68
ADF_Get_Root_ID	ADFGRID	Get the root ID of the ADF file	71

Data Query Routines

C	Fortran	Description	Page
ADF_Get_Label	ADFGLB	Get a label	73
ADF_Set_Label	ADFSLB	Set a label	75
ADF_Get_Data_Type	ADFGDT	Get the data type	76
ADF_Get_Number_of_Dimensions	ADFGND	Get the number of dimensions	79
ADF_Get_Dimension_Values	ADFGDV	Get the dimension values of a node	80
ADF_Put_Dimension_Information	ADFPDIM	Set the data type and dimension information	81

Data I/O Routines

C	Fortran	Description	Page
ADF_Read_Data	ADFREAD	Read the data from a node (with partial capabilities)	83
ADF_Read_All_Data	ADFRALL	Read all the data into a contiguous memory space	90
ADF_Read_Block_Data	ADFRBLK	Read a contiguous block of data from a node	91
ADF_Write_Data	ADFWRIT	Write the data to a node (with partial capabilities)	92
ADF_Write_All_Data	ADFWALL	Write all the data from a contiguous memory space	100
ADF_Write_Block_Data	ADFWBLK	Write a contiguous block of data to a node	101

I. 2 以字母排列的子程序列表

C	Fortran	Description	Page
ADF_Children_Names	ADFCNAM	Get the child names of a node	43
ADF_Create	ADFCRE	Create a node	35
ADF_Database_Close	ADFDCLO	Close an opened database	31
ADF_Database_Delete	ADFDDEL	Delete an existing database[7]	32
ADF_Database_Garbage_Collection	ADFDGC	Garbage collection[8]	103
ADF_Database_Get_Format	ADFDGF	Get the data format in existing database	33
ADF_Database_Open	ADFDOPN	Open a database	29
ADF_Database_Set_Format	ADFDSF	Set the data format in an existing database	34
ADF_Database_Version	ADFDVER	Get the ADF file version ID	109
ADF_Delete	ADFDEL	Delete a node	39
ADF_Error_Message	ADFERR	Return an error message	104
ADF_Flush_to_Disk	ADFFTD	Flush the data to a disk	102
ADF_Get_Data_Type	ADFGDT	Get the data type	76
ADF_Get_Dimension_Values	ADFGDV	Get the dimension values of a node	80
ADF_Get_Error_State	ADFGES	Get the error state	107
ADF_Get_Label	ADFGLB	Get a label	73
ADF_Get_Link_Path	ADFGLKP	Get the path information from a link	68
ADF_Get_Name	ADFGNAM	Get the name of a node	51
ADF_Get_Node_ID	ADFGNID	Get a unique identifier of a node	48
ADF_Get_Number_of_Dimensions	ADFGND	Get the number of dimensions	79
ADF_Get_Root_ID	ADFGRID	Get the root ID of the ADF file	71
ADF_Is_Link	ADFISLK	Test if a node is a link	66
ADF_Library_Version	ADFLVER	Get the ADF library version ID	111
ADF_Link	ADFLINK	Create a link	59
ADF_Move_Child	ADFMOVE	Change a parent (move a child)	55
ADF_Number_of_Children	ADFNCLD	Get the number of children of a node	47
ADF_Put_Dimension_Information	ADFPDIM	Set the data type and dimension information	81
ADF_Put_Name	ADFPNAM	Put (change) the name of a node	52
ADF_Read_All_Data	ADFRALL	Read all the data into a contiguous memory space	90
ADF_Read_Block_Data	ADFRBLK	Read a contiguous block of data from a node	91

C	Fortran	Description	Page
ADF_Read_Data	ADFREAD	Read the data from a node (with partial capabilities)	83
ADF_Set_Error_State	ADFSES	Set the error state	105
ADF_Set_Label	ADFSLB	Set a label	75
ADF_Write_All_Data	ADFWALL	Write all the data from a contiguous memory space	100
ADF_Write_Block_Data	ADFWBLK	Write a contiguous block of data to a node	101
ADF_Write_Data	ADFWRIT	Write the data to a node (with partial capabilities)	92

I. 3 Database-level 的子程序

ADF_Database_Open — *Open a Database*

ADF_Database_Open (filename,status,format,root_ID,error_return)

Language	C	Fortran
Routine Name	ADF_Database_Open	ADFDOPN
Input	const char *filename const char *status const char *format	character*(*) filename character*(*) status character*(*) format
Output	double *root_ID int *error_return	real*8 root_ID integer error_return

filename　一个包括相对的和绝对的路径的文件名，可以被 C 语言系统子程序 fopen()直接使用。

status　与 Fortran 中 open()的 status 相似。输入是需要的，没有默认值，允许的取值有以下几种：

READ_ONLY	File must exist; writing is not allowed.
OLD	File must exist; reading and writing are allowed.
NEW	File must not exist.
SCRATCH	Temporary new file is created with a system name, and *filename* is ignored. The temporary file is deleted when the program exits or the file is closed.
UNKNOWN	OLD if file exists or else NEW is used.

format　为文件指定数值格式。只有当文件被创建和抛弃（status=OLD）时，该区域才会用到，取值为：

NATIVE	Use the native numeric format of the computer that creates the file. This is the default for a new file if the input string for format is null. Note that if the native numeric format is not one of the supported formats listed here, then the file cannot be read on machines using any other format.
IEEE_BIG	Use the IEEE big endian format.
IEEE_LITTLE	Use the IEEE little endian format.
CRAY	Use the native CRAY format.

root_ID　数据库的根 ID。

error_return　错误返回码。（参见附录 G）

此子程序用来打开一个现存的或新的数据库。如果存在有连接到其他ADF文件的连接，只有在要求的情况下才打开。使用此子程序，与fortran中用open函数打开一个文件相同，需要阐述的是是否READ_ONLY,NEW,OLD或SEARCH文件。文件的格式在文件第一次被创建使用，当文件状态为Old时则被忽略。为了明确，可适用下列格式：

```
IEEE_BIG_32
IEEE_BIG_64
IEEE_LITTLE_32
IEEE_LITTLE_64
```

示例：

本例是用主机的那是格式打开一个新的数据库。注意默认的格式使用空字串来赋值。在C语言中，空字串是可以使用的。

```
PROGRAM TEST
   CHARACTER*(80) MSG
   REAL*8 RID
   INTEGER IERR
   CALL ADFDOPN('db.adf','NEW',' ',RID,IERR)
   IF (IERR .GT. 0) THEN
      CALL ADFERR(IERR,MSG)
      PRINT *,MSG
      STOP
   ENDIF
   STOP
   END
```

ADF_Database_Close — *Close a Database*

ADF_Database_Close (root_ID,error_return))

Language	C	Fortran
Routine Name	ADF_Database_Close	ADFDCLO
Input	const double *root_ID	real*8 root_ID
Output	int *error_return	integer error_return

root_ID　　数据库的根ID。

error_return　错误返回码。（参见附录G）

此子程序用来关闭现有的数据库和通过links连接的其他ADF文件。举例来说，如果其他

的ADF文件是打开的而且连接到这个数据库，只有与本数据库相连的文件才会被关闭。程序的动作与Fortran相同。

示例：

例中演示了关闭文件的过程。

```
PROGRAM TEST
   CHARACTER*(80) MSG
   REAL*8 RID
   INTEGER IERR
   CALL ADFDOPN('db.adf','NEW',' ',RID,IERR)
   IF (IERR .GT. 0) THEN
      CALL ADFERR(IERR,MSG)
      PRINT *,MSG
      STOP
   ENDIF
   .
   ...do useful stuff (hopefully)
   .
   CALL ADFDCLO(RID,IERR)
   STOP
   END
```

ADF_Database_Delete — *Delete a File*

ADF_Database_Delete (filename,error_return))

Language	C	Fortran
Routine Name	ADF_Database_Delete	ADFDDEL
Input	const char *filename	character*(*) filename
Output	int *error_return	integer error_return

filename　　一个包括相对的和绝对的路径的文件名，可以被C语言系统子程序fopen()直接使用。

error_return　错误返回码。(参见附录G)

此子程序用来删除现有的数据库文件。但不擅出指向数据库的连接（link），与Fortran中用法相同。

ADF_Database_Get_Format — *Get the Data Format*

ADF_Database_Get_Format (root_ID,format,error_return))

Language	C	Fortran
Routine Name	ADF_Database_Get_Format	ADFDGF
Input	const double *root_ID	real*8 root_ID
Output	char *format	character*(*) format
	int *error_return	integer error_return

root_ID　　数据库的根ID。

format　　文件格式。
error_return　错误返回码。(参见附录G)

此子程序用来提取文件的格式。

示例：

示例演示了打开一个已有的ADF数据库，创建一个新的结点,通过询问其ID得到其类型。由于数据库已然存在，所以格式被忽略了。

```
PROGRAM TEST
   CHARACTER*(80) MSG
   CHARACTER*(32) FORM
   REAL*8 RID,CID
   INTEGER IERR
   CALL ADFDOPN('db.adf','OLD',' ',RID,IERR)
   IF (IERR .GT. 0) THEN
      CALL ADFERR(IERR,MSG)
      PRINT *,MSG
      STOP
   ENDIF
   CALL ADFCRE(RID,'junk_node',CID,IERR)
   CALL ADFDGF(CID,FORM,IERR)
   PRINT *,'FILE FORMAT = ',FORM
   STOP
   END
```

ADF_Database_Set_Format — *Set the Data Format*

ADF_Database_Set_Format (root_ID,format,error_return))		
Language	C	Fortran
Routine Name	ADF_Database_Set_Format	ADFDSF
Input	const double *root_ID	real*8 root_ID
Output	char *format	character*(*) format
	int *error_return	integer error_return

此程序用来设置已有的数据库的数据格式。
特别要注意：此程序只能被数据转化工具使用，而非一般用户。

I.4 数据结构和管理子程序

ADF_Create — *Create a Node*

ADF_Create (PID,name,ID,error_return)

Language	C	Fortran
Routine Name	ADF_Create	ADFCRE
Input	const double PID	real*8 PID
	const char *name	character*(*) name
Output	double *ID	real*8 ID
	int *error_return	integer error_return

PID　　创建子结点的父结点的ID。

此子程序用来为指定的父结点创建一个新的子结点。

新结点头部的取值如下：

- label = blank
- number of subnodes = 0
- datatype = MT
- number of dimensions = 0
- data = NULL

示例：

```
PROGRAM TEST
C
      PARAMETER (MAXCHR=32)
C
      CHARACTER*(MAXCHR) NODNAM
C
C *** NODE IDS
C
      REAL*8 RID,PID,CID
      INTEGER I,J,IERR,NUMCLD
```

```
C
C *** OPEN DATABASE
C
      CALL ADFDOPN('db.adf','NEW',' ',RID,IERR)
C
C *** CREATE NODES AT FIRST LEVEL
C
      DO 150 I = 1,3
         WRITE(NODNAM,'(A7,I1)')'PARENT.',I
         CALL ADFCRE(RID,NODNAM,PID,IERR)
C
C ****** CREATE NODES AT SECOND LEVEL
C
         NUMCLD = I*I
         DO 110 J = 1,NUMCLD
            WRITE(NODNAM,'(A6,I1,A1,I1)')'CHILD.',I,'x',J
            CALL ADFCRE(PID,NODNAM,CID,IERR)
  110    CONTINUE
C
C ****** PRINT NODE NAMES JUST CREATED
C
         CALL PRTCLD(PID)
  150 CONTINUE
C
C *** PRINT NAMES OF NODES ATTACHED TO ROOT NODE
C
      CALL PRTCLD(RID)
C
      STOP
      END
C
C ************* SUBROUTINES ****************
C
      SUBROUTINE ERRCHK(IERR)
C
C *** CHECK ERROR CONDITION
C
      CHARACTER*80 MESS
      IF (IERR .GT. 0) THEN
         CALL ADFERR(IERR,MESS)
         PRINT *,MESS
         CALL ABORT('ADF ERROR')
      ENDIF
      RETURN
      END

      SUBROUTINE PRTCLD(PID)
C
C *** PRINT TABLE OF CHILDREN GIVEN A PARENT NODE-ID
C
      PARAMETER (MAXCLD=10)
```

```
      PARAMETER (MAXCHR=32)
      REAL*8 PID
      CHARACTER*(MAXCHR) NODNAM,NDNMS(MAXCLD)
      CALL ADFGNAM(PID,NODNAM,IERR)
      CALL ERRCHK(IERR)
      CALL ADFNCLD(PID,NUMC,IERR)
      CALL ERRCHK(IERR)
      WRITE(*,120)NODNAM,NUMC
  120 FORMAT(/,' PARENT NODE NAME = ',A,/,
     X       '     NUMBER OF CHILDREN = ',I2,/,
     X       '     CHILDREN NAMES: ')
      NLEFT = NUMC
      ISTART = 1
C     --- TOP OF DO-WHILE LOOP
  130 CONTINUE
        CALL ADFCNAM(PID,ISTART,MAXCLD,LEN(NDNMS),
     X               NUMRET,NDNMS,IERR)
        CALL ERRCHK(IERR)
        WRITE(*,140)(NDNMS(K),K=1,NUMRET)
  140   FORMAT(2(8X,A))
        NLEFT = NLEFT - MAXCLD
        ISTART = ISTART + MAXCLD
      IF (NLEFT .GT. 0) GO TO 130
      RETURN
      END
```

The resulting output is:

```
PARENT NODE NAME = PARENT.1
    NUMBER OF CHILDREN =  1
    CHILDREN NAMES:
       CHILD.1.1

PARENT NODE NAME = PARENT.2
    NUMBER OF CHILDREN =  4
    CHILDREN NAMES:
       CHILD.2.1                                CHILD.2.2
       CHILD.2.3                                CHILD.2.4

PARENT NODE NAME = PARENT.3
    NUMBER OF CHILDREN =  9
    CHILDREN NAMES:
       CHILD.3.1                                CHILD.3.2
       CHILD.3.3                                CHILD.3.4
       CHILD.3.5                                CHILD.3.6
       CHILD.3.7                                CHILD.3.8
       CHILD.3.9

PARENT NODE NAME = ADF MotherNode
    NUMBER OF CHILDREN =  3
    CHILDREN NAMES:
```

```
PARENT.1                                PARENT.2
PARENT.3
```

ADF_Delete — *Delete a Node*

ADF_Delete (PID,ID,error_return)		
Language	**C**	**Fortran**
Routine Name	ADF_Delete	ADFDEL
Input	const double PID	real*8 PID
	const double ID	real*8 ID
Output	int *error_return	integer error_return

一般来说，此子程序用来删除一个结点和它所有的子结点。给定一个起始的结点，一个递归向下对分层进行的搜索就启动了，然后就是删除所有的结点。在这个过程中，如果遇到一个连接，则连接被删除，向下搜索停止。就是连接被删除，而不是所指向的结点被删除。

示例：

```
PROGRAM TEST
C
      PARAMETER (MAXCHR=32)
C
      CHARACTER*(MAXCHR) NODNAM
C
C *** NODE IDS
C
      REAL*8 RID,PID,CID
```

```
      INTEGER I,J,IERR,NUMCLD
C
C *** OPEN DATABASE
C
      CALL ADFDOPN('db.adf','NEW',' ',RID,IERR)
C
C *** CREATE NODES AT FIRST LEVEL
C
      DO 150 I = 1,3
         WRITE(NODNAM,'(A7,I1)')'PARENT.',I
         CALL ADFCRE(RID,NODNAM,PID,IERR)
C
C ****** CREATE NODES AT SECOND LEVEL
C
         NUMCLD = I*I
         DO 110 J = 1,NUMCLD
            WRITE(NODNAM,'(A6,I1,A1,I1)')'CHILD.',I,'.',J
            CALL ADFCRE(PID,NODNAM,CID,IERR)
  110    CONTINUE
  150 CONTINUE
C
C *** PRINT NAMES OF NODES ATTACHED TO ROOT NODE
C
      CALL PRTCLD(RID)
C
C *** PRINT NAMES OF CHILDREN UNDER PARENT.2
C
      CALL ADFGNID(RID,'PARENT.2',PID,IERR)
      CALL PRTCLD(PID)
C
C *** NOW DELETE PARENT.2
C
      CALL ADFDEL(RID,PID,IERR)
      CALL PRTCLD(RID)
C
C *** JUST FOR GRINS, LOOK FOR CHILDREN UNDER ORIGINAL ID
C
      CALL ADFGNID(RID,'/PARENT.2/CHILD.2.1',CID,IERR)
      CALL ERRCHK(IERR)
C
      STOP
      END
C
C ************* SUBROUTINES ****************
C
      SUBROUTINE ERRCHK(IERR)
C
C *** CHECK ERROR CONDITION
C
      CHARACTER*80 MESS
      IF (IERR .GT. 0) THEN
```

```
            CALL ADFERR(IERR,MESS)
            PRINT *,MESS
            CALL ABORT('ADF ERROR')
         ENDIF
         RETURN
         END

         SUBROUTINE PRTCLD(PID)
C
C *** PRINT TABLE OF CHILDREN GIVEN A PARENT NODE-ID
C
         PARAMETER (MAXCLD=10)
         PARAMETER (MAXCHR=32)
         REAL*8 PID
         CHARACTER*(MAXCHR) NODNAM,NDNMS(MAXCLD)
         CALL ADFGNAM(PID,NODNAM,IERR)
         CALL ERRCHK(IERR)
         CALL ADFNCLD(PID,NUMC,IERR)
         CALL ERRCHK(IERR)
         WRITE(*,120)NODNAM,NUMC
  120 FORMAT(/,' PARENT NODE NAME = ',A,/,
        X       '     NUMBER OF CHILDREN = ',I2,/,
        X       '     CHILDREN NAMES:')
         NLEFT = NUMC
         ISTART = 1
C        --- TOP OF DO-WHILE LOOP
  130 CONTINUE
            CALL ADFCNAM(PID,ISTART,MAXCLD,LEN(NDNMS),
        X                NUMRET,NDNMS,IERR)
            CALL ERRCHK(IERR)
            WRITE(*,140)(NDNMS(K),K=1,NUMRET)
  140    FORMAT(2(8X,A))
            NLEFT = NLEFT - MAXCLD
            ISTART = ISTART + MAXCLD
         IF (NLEFT .GT. 0) GO TO 130
         RETURN
         END
```

The resulting output is:

```
PARENT NODE NAME = ADF MotherNode
    NUMBER OF CHILDREN = 3
    CHILDREN NAMES:
       PARENT.1                                PARENT.2
       PARENT.3

PARENT NODE NAME = PARENT.2
    NUMBER OF CHILDREN = 4
    CHILDREN NAMES:
       CHILD.2.1                               CHILD.2.2
       CHILD.2.3                               CHILD.2.4
```

```
PARENT NODE NAME = ADF MotherNode
    NUMBER OF CHILDREN = 2
    CHILDREN NAMES:
       PARENT.1                                  PARENT.3

 ADF 29: Specified child is NOT a child of the specified parent.
IOT Trap
Abort - core dumped
```

ADF_Children_Names — *Get the Names of the Child Nodes*

```
ADF_Children_Names (PID,istart,imax_num,imax_name_length,inum_ret,names,
                    error_return)
```

Language Routine Name	C ADF_Children_Names	Fortran ADFCNAM
Input	const double PID	real*8 PID
	const int istart	integer istart
	const int imax_num	integer imax_num
	const int imax_name_length	integer imax_name_length
Output	int *inum_ret	integer inum_ret
	char *names	character*(*) names
	int *error_return	integer error_return

PID	The ID of the parent node to use.
istart	The *n*th child's name (to start with the first, use *istart* = 1).
imax_num	The maximum number of names to return.
imax_name_length	The number of characters allocated to hold the name of each child node.
inum_ret	The number of names returned.
names	The names of the children.
error_return	Error return code. (See Appendix G.)

此程序用来返回父结点的子结点的names，返回的子结点的names并不是以某一特定的顺序排列。比如，按顺序生成的四个子结点node1, node2, node3, node4, 调用ADFCNAM, 并不能保证四个结点的名称按相同的顺序出现在名称数组中。

不按顺是为了保证磁盘的空间使用效率考虑。尽管连接表（linked list）在中央内存中表现良好，但并不意味着在磁盘上表现的有效率。因此，静态表格（static tables）用来维持父/子结点列表。调用返回的子结点名称顺序可在静态表中找到。如果一个子结点被删除，会产生一个空的格位（slot）没，用来存放下一个父结点生成的子结点名称。

I. C语言中编程需注意的事项：

- 结点的字符数可以达到32个。ADF需要在最后附上空字符（\0）。因此，返回的结点字符数为33个；
- imax name length用来大幅度的把字符数组传递给ADF。比如，每个名称（name）设置了50个字符，那么第一个名称的第一个字符放置在第一个字节（byte）处[position(0)]，第二个名称的第一个字符放置在堤1个字节处[position(50)],

以此类推。可以用strlen函数来决定单个名称的长度。

II. Fortran语言编程需注意的事项：

- 结点字符数可达32个。返回的名称在数组中向左排列并进行空白填充。不能把空字符（null）附到名称的后面；因此，正确的定义应该是CHARACTER*(32)。

示例：

```
      PROGRAM TEST
C
      PARAMETER (MAXCLD=5)
      PARAMETER (NDATA=10)
      PARAMETER (MAXCHR=32)
C
      CHARACTER*(MAXCHR) NODNAM,NDNMS(MAXCLD)
C
C     RID - ROOT ID
C     CID - CHILD ID
C     PID - PARENT ID
C
      REAL*8 RID,CID,PID
      INTEGER I,J,K,IERR,NUMCLD,NLEFT,ISTART
C
C *** OPEN DATABASE
C
      CALL ADFDOPN('DB.ADF','NEW',' ',RID,IERR)
      CALL ERRCHK(IERR)
C
C *** CREATE NODES AT FIRST LEVEL
C
      DO 150 I = 1,3
         WRITE(NODNAM,'(A7,I1)')'PARENT.',I
         CALL ADFCRE(RID,NODNAM,PID,IERR)
         CALL ERRCHK(IERR)
C
C ****** CREATE NODES AT SECOND LEVEL
```

```
C
         NUMCLD = I*I
         DO 110 J = 1,NUMCLD
            WRITE(NODNAM,'(A6,I1,A1,I1)')'CHILD.',I,'.',J
            CALL ADFCRE(PID,NODNAM,CID,IERR)
            CALL ERRCHK(IERR)
  110    CONTINUE
C
C ****** GET NUMBER AND NAMES OF CHILDREN JUST CREATED
C        AND PRINT THEM OUT
C
         CALL ADFGNAM(PID,NODNAM,IERR)
         CALL ERRCHK(IERR)
         CALL ADFNCLD(PID,NUMC,IERR)
         CALL ERRCHK(IERR)
         WRITE(*,120)I,NODNAM,NUMC
  120    FORMAT(' LEVEL = ',I2,' PARENT NODE NAME = ',A,/,
     X          '     NUMBER OF CHILDREN = ',I2,/,
     X          '     CHILDREN NAMES:')
         NLEFT = NUMC
         ISTART = 1
C        --- TOP OF DO-WHILE LOOP
  130    CONTINUE
            CALL ADFCNAM(PID,ISTART,MAXCLD,LEN(NDNMS),
     X                   NUMRET,NDNMS,IERR)
            CALL ERRCHK(IERR)
            PRINT *,' FETCHED: ',NUMRET,' NAMES'
            WRITE(*,140)(NDNMS(K),K=1,NUMRET)
  140       FORMAT(8X,A)
            NLEFT = NLEFT - MAXCLD
            ISTART = ISTART + MAXCLD
         IF (NLEFT .GT. 0) GO TO 130
  150 CONTINUE
      STOP
      END
C
C ************** SUBROUTINES *****************
C
      SUBROUTINE ERRCHK(IERR)
      CHARACTER*80 MESS
      IF (IERR .GT. 0) THEN
         CALL ADFERR(IERR,MESS)
         PRINT *,MESS
         CALL ABORT('ADF ERROR')
      ENDIF
      RETURN
      END
```

The resulting output is:

```
 PARENT = 1 PARENT NODE NAME = PARENT.1
```

```
    NUMBER OF CHILDREN =  1
    CHILDREN NAMES:
     FETCHED            1 NAMES
       CHILD.1.1

PARENT = 2 PARENT NODE NAME = PARENT.2
    NUMBER OF CHILDREN =  4
    CHILDREN NAMES:
     FETCHED            4 NAMES
       CHILD.2.1
       CHILD.2.2
       CHILD.2.3
       CHILD.2.4

PARENT = 3 PARENT NODE NAME = PARENT.3
    NUMBER OF CHILDREN =  9
    CHILDREN NAMES:
     FETCHED            5 NAMES
       CHILD.3.1
       CHILD.3.2
       CHILD.3.3
       CHILD.3.4
       CHILD.3.5
     FETCHED            4 NAMES
       CHILD.3.6
       CHILD.3.7
       CHILD.3.8
       CHILD.3.9
```

ADF_Number_of_Children — *Get the Number of Children Nodes*

ADF_Number_of_Children (PID,num_children,error_return)

Language	C	Fortran
Routine Name	ADF_Number_of_Children	ADFNCLD
Input	const double PID	real*8 PID
Output	int *num_children	integer num_children
	int *error_return	integer error_return

PID The ID of the parent node to use.

num_children The number of children directly associated with this node.

error_return Error return code. (See Appendix G.)

此子程序调用返回父结点的子结点的数量。

示例：见 ADF_Children_Names中的示例。

ADF_Get_Node_ID — *Get the ID of a Child Node*

ADF_Get_Node_ID (PID,name,ID,error_return)		
Language	**C**	**Fortran**
Routine Name	ADF_Get_Node_ID	ADFGNID
Input	const double PID	real*8 PID
	const char *name	character*(*) name
Output	double *ID	real*8 ID
	int *error_return	integer error_return

PID The ID of the parent node.

name The name of the node.

ID The ID of the named node.

error_return Error return code. (See Appendix G.)

此子程序用调用返回给定的父结点和子结点的ID。要返回ADF文件根结点的ID，可以使用ADF文件中的已知结点或 “/”。在UNIX操作系统中，name的语法与path name本质上相同，比如/level.1/level.2/node。结点名称为其中之一。如果名称以“/”开头，名称与数据库根结点的名称相关。如果不是，则与父结点的PID有关。

示例：

```
PROGRAM TEST
C
      PARAMETER (MAXCHR=32)
C
      CHARACTER*(MAXCHR) NODNAM
C
C     RID - ROOT ID
C     AL1ID - LEVEL 1 ID
C     AL2ID - LEVEL 2 ID
C     AL3ID - LEVEL 3 ID
C     CID - CHILD ID
C
      REAL*8 RID,AL1ID,AL2ID,AL3ID,CID
```

```
C
C *** OPEN DATABASE
C
      CALL ADFDOPN('DB.ADF','NEW',' ',RID,IERR)
      CALL ERRCHK(IERR)
C
C *** CREATE NODE AT FIRST LEVEL
C
      CALL ADFCRE(RID,'LEVEL.1',AL1ID,IERR)
      CALL ERRCHK(IERR)
C
C *** CREATE NODE AT SECOND LEVEL
C
      CALL ADFCRE(AL1ID,'LEVEL.2',AL2ID,IERR)
      CALL ERRCHK(IERR)
C
C *** CREATE NODE AT THIRD LEVEL
C
      CALL ADFCRE(AL2ID,'LEVEL.3',AL3ID,IERR)
      CALL ERRCHK(IERR)
C
C *** EQUIVALENT WAYS TO GET THE LOWER LEVEL NODE ID
C
C ****** FULL PATH NAME
C
      CALL ADFGNID(RID,'/LEVEL.1/LEVEL.2/LEVEL.3',CID,IERR)
      CALL ERRCHK(IERR)
      CALL ADFGNAM(CID,NODNAM,IERR)
      PRINT *,' '
      PRINT *,' FULL PATH EXAMPLE: ROOT NODE ID: NODE NAME = ',NODNAM
C
C ****** FULL PATH NAME - GIVEN ANY VALID NODE ID FOR FILE
C
      CALL ADFGNID(AL3ID,'/LEVEL.1/LEVEL.2/LEVEL.3',CID,IERR)
      CALL ERRCHK(IERR)
      CALL ADFGNAM(CID,NODNAM,IERR)
      PRINT *,' '
      PRINT *,' FULL PATH EXAMPLE - VALID NODE ID: NODE NAME = ',NODNAM
C
C ****** PARTIAL PATH NAME
C
      CALL ADFGNID(AL1ID,'LEVEL.2/LEVEL.3',CID,IERR)
      CALL ERRCHK(IERR)
      CALL ADFGNAM(CID,NODNAM,IERR)
      PRINT *,' '
      PRINT *,' PARTIAL PATH EXAMPLE: NODE NAME = ',NODNAM
C
C ****** DIRECT USE OF PARENT ID
C
      CALL ADFGNID(AL2ID,'LEVEL.3',CID,IERR)
      CALL ERRCHK(IERR)
```

```
      CALL ADFGNAM(CID,NODNAM,IERR)
      PRINT *,' '
      PRINT *,' GIVEN PARENT NAME EXAMPLE: NODE NAME = ',NODNAM
      STOP
      END
C
C ************** SUBROUTINES *****************
C
      SUBROUTINE ERRCHK(IERR)
      CHARACTER*80 MESS
      IF (IERR .GT. 0) THEN
         CALL ADFERR(IERR,MESS)
         PRINT *,MESS
         CALL ABORT('ADF ERROR')
      ENDIF
      RETURN
      END
```

The resulting output is:

```
FULL PATH EXAMPLE: ROOT NODE ID: NODE NAME = LEVEL.3

FULL PATH EXAMPLE - VALID NODE ID: NODE NAME = LEVEL.3

PARTIAL PATH EXAMPLE: NODE NAME = LEVEL.3

GIVEN PARENT NAME EXAMPLE: NODE NAME = LEVEL.3
```

ADF_Get_Name — *Get the Name of a Node*

ADF_Get_Name (ID,name,error_return)		
Language	**C**	**Fortran**
Routine Name	ADF_Get_Name	ADFGNAM
Input	const double ID	real*8 ID
Output	char *name	character*(*) name
	int *error_return	integer error_return

ID The ID of the node to use.

name The simple name of the node.

error_return Error return code. (See Appendix G.)

此子程序用来调用返回一个结点的32字符名称。在C语言中，非空字符后一空字符结束，所以应该为33字符。

示例：见ADF_Get_Node_ID中的示例。

ADF_Put_Name — *Put a Name on a Node*

ADF_Put_Name (PID,ID,name,error_return)		
Language	**C**	**Fortran**
Routine Name	ADF_Put_Name	ADFPNAM
Input	const double PID const double ID const char *name	real*8 PID real*8 ID character*(*) name
Output	int *error_return	integer error_return

PID The ID of the node's parent.

ID The ID of the node to use.

name The new name of the node.

error_return Error return code. (See Appendix G.)

此子程序用来更改一个结点的名称。

警告：当一个连接结点（link node）指向结点，更改结点的名称时连接结点将会被破坏。

示例：

```
      PROGRAM TEST
C
            PARAMETER (MAXCHR=32)
C
            CHARACTER*(MAXCHR) NODNAM,CLDNAM
C
C           RID - ROOT ID
C           CID - CHILD ID
C
            REAL*8 RID,AL1ID,AL2ID,AL3ID,CID
C
C *** OPEN DATABASE
C
            CALL ADFDOPN('db.adf','NEW',' ',RID,IERR)
C
            CALL ERRCHK(IERR)
C
C *** CREATE NODE
```

```
C
      CALL ADFCRE(RID,'LEVEL.1',CID,IERR)
      CALL ERRCHK(IERR)
C
C *** GET NODE NAME AND CHECK PARENTS TABLE
C
      CALL ADFGNAM(CID,NODNAM,IERR)
      CALL ERRCHK(IERR)
      PRINT *,' '
      PRINT *,' NODE NAME = ',NODNAM
      CALL ADFCNAM(RID,1,1,LEN(CLDNAM),
     X             NUMRET,CLDNAM,IERR)
      CALL ERRCHK(IERR)
      PRINT *,' NODE NAME IN PARENTS TABLE = ',CLDNAM
C
C *** CHANGE THE NODE NAME
C
      CALL ADFPNAM(RID,CID,'NEW_NAME',IERR)
      CALL ERRCHK(IERR)
C
C *** GET NEW NODE NAME AND CHECK PARENTS TABLE
C
      CALL ADFGNAM(CID,NODNAM,IERR)
      CALL ERRCHK(IERR)
      PRINT *,' '
      PRINT *,' NEW NODE NAME = ',NODNAM
      CALL ADFCNAM(RID,1,1,LEN(CLDNAM),
     X             NUMRET,CLDNAM,IERR)
      CALL ERRCHK(IERR)
      PRINT *,' NODE NAME IN PARENTS TABLE = ',CLDNAM
      STOP
      END
C
C ************* SUBROUTINES ****************
C
      SUBROUTINE ERRCHK(IERR)
      CHARACTER*80 MESS
      IF (IERR .GT. 0) THEN
         CALL ADFERR(IERR,MESS)
         PRINT *,MESS
         CALL ABORT('ADF ERROR')
      ENDIF
      RETURN
      END
```

The resulting output is:

```
 NODE NAME     = LEVEL.1
 NODE NAME IN PARENTS TABLE = LEVEL.1

 NEW NODE NAME = NEW_NAME

NODE NAME IN PARENTS TABLE = NEW_NAME
```

ADF_Move_Child — *Move a Child Node to a Different Parent*

ADF_Move_Child (PID,ID,NPID,error_return)

Language	C	Fortran
Routine Name	ADF_Move_Child	ADFMOVE
Input	const double PID	real*8 PID
	const double ID	real*8 ID
	double NPID	real*8 NPID
Output	int *error_return	integer error_return

PID ID of the node's current parent.

ID ID of the node to use.

NPID ID of the node's new parent.

error_return Error return code. (See Appendix G.)

此子程序用来删除来自父结点的子结点表中的一个子结点，并将该子结点加到一个新的父结点的子结点表中。该动作严格限制在物理上相同的文件中。

示例：

```
      PROGRAM TEST
C
      PARAMETER (MAXCHR=32)
C
      CHARACTER*(MAXCHR) NODNAM
C
C *** NODE IDS
C
      REAL*8 RID,PID,CID,PID1,PID3
      INTEGER I,J,IERR
C
C *** OPEN DATABASE
C
      CALL ADFDOPN('db.adf','NEW',' ',RID,IERR)
      CALL ERRCHK(IERR)
C
C *** CREATE NODES AT FIRST LEVEL
C
      WRITE(*,100)
  100 FORMAT(/,' *** ORIGINAL DATABASE SETUP ***')
      DO 150 I = 1,3
```

```
         WRITE(NODNAM,'(A7,I1)')'PARENT.',I
         CALL ADFCRE(RID,NODNAM,PID,IERR)
         CALL ERRCHK(IERR)
C
C ****** CREATE NODES AT SECOND LEVEL
C
         NUMCLD = I*I
         DO 110 J = 1,NUMCLD
            WRITE(NODNAM,'(A6,I1,A1,I1)')'CHILD.',I,'.',J
            CALL ADFCRE(PID,NODNAM,CID,IERR)
            CALL ERRCHK(IERR)
  110    CONTINUE
C
C ****** GET NUMBER AND NAMES OF CHILDREN JUST CREATED
C        AND PRINT THEM OUT
C
         CALL PRTCLD(PID)
  150 CONTINUE
C
C *** PICK UP NODE /PARENT.3/CHILD.3.4 AND MOVE IT
C     TO /PARENT.1
C
      CALL ADFGNID(RID,'PARENT.3',PID3,IERR)
      CALL ERRCHK(IERR)
C
      CALL ADFGNID(PID3,'CHILD.3.4',CID,IERR)
      CALL ERRCHK(IERR)
C
      CALL ADFGNID(RID,'PARENT.1',PID1,IERR)
      CALL ERRCHK(IERR)
C
      CALL ADFMOVE(PID3,CID,PID1,IERR)
      CALL ERRCHK(IERR)
C
C *** CHECK TO MAKE SURE THE NODE WAS ACTUALLY MOVED
C
      WRITE(*,160)
  160 FORMAT(/,'*** PARENT.1 AND PARENT.3 AFTER MOVE ***')
      CALL PRTCLD(PID1)
      CALL PRTCLD(PID3)
C
      STOP
      END
C
C ************** SUBROUTINES *****************
C
      SUBROUTINE ERRCHK(IERR)
C
C *** CHECK ERROR CONDITION
C
      CHARACTER*80 MESS
```

```
      IF (IERR .GT. 0) THEN
         CALL ADFERR(IERR,MESS)
         PRINT *,MESS
         CALL ABORT('ADF ERROR')
      ENDIF
      RETURN
      END

      SUBROUTINE PRTCLD(PID)
C
C *** PRINT TABLE OF CHILDREN GIVEN A PARENT NODE-ID
C
      PARAMETER (MAXCLD=10)
      PARAMETER (MAXCHR=32)
      REAL*8 PID
      CHARACTER*(MAXCHR) NODNAM,NDNMS(MAXCLD)
      CALL ADFGNAM(PID,NODNAM,IERR)
      CALL ERRCHK(IERR)
      CALL ADFNCLD(PID,NUMC,IERR)
      CALL ERRCHK(IERR)
      WRITE(*,120)NODNAM,NUMC
  120 FORMAT(/,' PARENT NODE NAME = ',A,/,
     X       '     NUMBER OF CHILDREN = ',I2,/,
     X       '     CHILDREN NAMES:')
      NLEFT = NUMC
      ISTART = 1
C     === TOP OF DO-WHILE LOOP
  130 CONTINUE
         CALL ADFCNAM(PID,ISTART,MAXCLD,LEN(NDNMS),
     X                NUMRET,NDNMS,IERR)
         CALL ERRCHK(IERR)
         WRITE(*,140)(NDNMS(K),K=1,NUMRET)
  140    FORMAT(2(8X,A))
         NLEFT = NLEFT - MAXCLD
         ISTART = ISTART + MAXCLD
      IF (NLEFT .GT. 0) GO TO 130
      RETURN
      END
```

The resulting output is:

```
*** ORIGINAL DATABASE SETUP ***

PARENT NODE NAME = PARENT.1
    NUMBER OF CHILDREN =  1
    CHILDREN NAMES:
      CHILD.1.1

PARENT NODE NAME = PARENT.2
    NUMBER OF CHILDREN =  4
    CHILDREN NAMES:
```

```
       CHILD.2.1                      CHILD.2.2
       CHILD.2.3                      CHILD.2.4

PARENT NODE NAME = PARENT.3
    NUMBER OF CHILDREN =  9
    CHILDREN NAMES:
       CHILD.3.1                      CHILD.3.2
       CHILD.3.3                      CHILD.3.4
       CHILD.3.5                      CHILD.3.6
       CHILD.3.7                      CHILD.3.8
       CHILD.3.9

*** PARENT.1 AND PARENT.3 AFTER MOVE ***

PARENT NODE NAME = PARENT.1
    NUMBER OF CHILDREN =  2
    CHILDREN NAMES:
       CHILD.1.1                      CHILD.3.4

PARENT NODE NAME = PARENT.3
    NUMBER OF CHILDREN =  8
    CHILDREN NAMES:
       CHILD.3.1                      CHILD.3.2
       CHILD.3.3                      CHILD.3.5
       CHILD.3.6                      CHILD.3.7
       CHILD.3.8                      CHILD.3.9
```

ADF_Link — *Create a Link to a Node*

ADF_Link (PID,name,file,name_in_file,ID,error_return)

Language	C	Fortran
Routine Name	ADF_Link	ADFLINK
Input	const double PID	real*8 PID
	const char *name	character*(*) name
	const char *file	character*(*) file
	const char *name_in_file	character*(*) name_in_file
Output	double ID	real*8 ID
	int *error_return	integer error_return

Name_in_file link指向的结点名称，可以是单个结点也可是复合型结点。

此子程序用来创建一个指向ADF数据库文件或另外的ADF数据库文件中的结点的连接。当连接（link）创建时，并不保证连接结点指向的结点一定存在。当ADF_link被调用时，ADF库并不检查指向的结点是否存在。然而，当ADF_Get_Label和ADF_Read_Data等函数调用请求连接的结点信息时，连接的结点是可用的。如果不存在，则返回错误。ADF_Link与UNIX中“soft link”相似。

示例1：

```
      PROGRAM TEST
      C
            PARAMETER (MAXCHR=32)
      C
            CHARACTER*(MAXCHR) NODNAM,NODLBL,TSTLBL
            CHARACTER*(72) FN,PATH
      C
      C *** NODE IDS
```


原创力文档
max.book118.com
预览与源文档一致,下载高清无水印

原创力文档
max.book118.com
预览与源文档一致,下载高清无水印

```
C
      REAL*8 RID,PID,CID,PID1,PID3
      INTEGER I,J,IERR
C
C *** OPEN DATABASE
C
      CALL ADFDOPN('db.adf','NEW',' ',RID,IERR)
      CALL ERRCHK(IERR)
C
C *** CREATE NODES AT FIRST LEVEL
C
      WRITE(*,100)
  100 FORMAT(/,' *** ORIGINAL DATABASE SETUP ***')
      DO 150 I = 1,3
         WRITE(NODNAM,'(A7,I1)')'PARENT.',I
         CALL ADFCRE(RID,NODNAM,PID,IERR)
         CALL ERRCHK(IERR)
C
C ****** CREATE NODES AT SECOND LEVEL
C
         NUMCLD = I*I
         DO 110 J = 1,NUMCLD
            WRITE(NODNAM,'(A6,I1,A1,I1)')'CHILD.',I,'.',J
            CALL ADFCRE(PID,NODNAM,CID,IERR)
            CALL ERRCHK(IERR)
            WRITE(NODLBL,105)I,J
  105       FORMAT('LABEL STRING IN CHILD.',I1,'.',I1)
            CALL ADFSLB(CID,NODLBL,IERR)
            CALL ERRCHK(IERR)
  110    CONTINUE
C
C ****** GET NUMBER AND NAMES OF CHILDREN JUST CREATED
C        AND PRINT THEM OUT
C
         CALL PRTCLD(PID)
  150 CONTINUE
C
C *** LINK NODE /PARENT.3/CHILD.3.4 TO /PARENT.1
C
      CALL ADFGNID(RID,'PARENT.1',PID1,IERR)
      CALL ERRCHK(IERR)
C
      CALL ADFLINK(PID1,'LINKED_NODE',' ',
     X             '/PARENT.3/CHILD.3.4',CID,IERR)
      CALL ERRCHK(IERR)
C
C *** CHECK TO MAKE SURE THE NODE WAS ACTUALLY LINKED
C
      WRITE(*,160)
  160 FORMAT(/,'*** PARENT.1 AFTER LINK ***')
      CALL PRTCLD(PID1)
```

```
C
C *** FOR FINAL CONFIRMATION, GET ORIGINAL LABEL
C     GOING THROUGH NEW LINK
C
      CALL ADFGLB(CID,TSTLBL,IERR)
      CALL ERRCHK(IERR)
      WRITE(*,170)TSTLBL
  170 FORMAT(/,'LINKED_NODE LABEL = ',A)
C
      STOP
      END
C
C ************* SUBROUTINES ****************
C
      SUBROUTINE ERRCHK(IERR)
C
C *** CHECK ERROR CONDITION
C
      CHARACTER*80 MESS
      IF (IERR .GT. 0) THEN
         CALL ADFERR(IERR,MESS)
         PRINT *,MESS
         CALL ABORT('ADF ERROR')
      ENDIF
      RETURN
      END

      SUBROUTINE PRTCLD(PID)
C
C *** PRINT TABLE OF CHILDREN GIVEN A PARENT NODE-ID
C
      PARAMETER (MAXCLD=10)
      PARAMETER (MAXCHR=32)
      REAL*8 PID
      CHARACTER*(MAXCHR) NODNAM,NDNMS(MAXCLD)
      CALL ADFGNAM(PID,NODNAM,IERR)
      CALL ERRCHK(IERR)
      CALL ADFNCLD(PID,NUMC,IERR)
      CALL ERRCHK(IERR)
      WRITE(*,120)NODNAM,NUMC
  120 FORMAT(/,' PARENT NODE NAME = ',A,/,
     X       '     NUMBER OF CHILDREN = ',I2,/,
     X       '     CHILDREN NAMES:')
      NLEFT = NUMC
      ISTART = 1
C     --- TOP OF DO-WHILE LOOP
  130 CONTINUE
         CALL ADFCNAM(PID,ISTART,MAXCLD,LEN(NDNMS),
     X                NUMRET,NDNMS,IERR)
         CALL ERRCHK(IERR)
         WRITE(*,140)(NDNMS(K),K=1,NUMRET)
```

```
  140    FORMAT(2(8X,A))
         NLEFT = NLEFT - MAXCLD
         ISTART = ISTART + MAXCLD
       IF (NLEFT .GT. 0) GO TO 130
       RETURN
       END
```

The resulting output is:

```
*** ORIGINAL DATABASE SETUP ***

PARENT NODE NAME = PARENT.1
    NUMBER OF CHILDREN =  1
    CHILDREN NAMES:
       CHILD.1.1

PARENT NODE NAME = PARENT.2
    NUMBER OF CHILDREN =  4
    CHILDREN NAMES:
       CHILD.2.1                      CHILD.2.2
       CHILD.2.3                      CHILD.2.4

PARENT NODE NAME = PARENT.3
    NUMBER OF CHILDREN =  9
    CHILDREN NAMES:
       CHILD.3.1                      CHILD.3.2
       CHILD.3.3                      CHILD.3.4
       CHILD.3.5                      CHILD.3.6
       CHILD.3.7                      CHILD.3.8
       CHILD.3.9

*** PARENT.1 AFTER LINK ***

PARENT NODE NAME = PARENT.1
    NUMBER OF CHILDREN =  2
    CHILDREN NAMES:
       CHILD.1.1                      LINKED_NODE

LINKED_NODE LABEL = LABEL STRING IN CHILD.3.4
```

Example 2

This example illustrates the linking of nodes across files.

```
PROGRAM TEST
C
      PARAMETER (MAXCHR=32)
C
      CHARACTER*(MAXCHR) TSTLBL
C
```

```
C *** NODE IDS
C
      REAL*8 RID,PID,CID
      INTEGER IERR
C
C *** 1.) OPEN 1ST DATABASE
C     2.) CREATE TWO NODES
C     3.) PUT LABEL ON 2ND NODE
C     4.) CLOSE DATABASE
C
      CALL ADFDOPN('db1.adf','NEW',' ',RID,IERR)
      CALL ERRCHK(IERR)
      CALL ADFCRE(RID,'DB1_NODE1',PID,IERR)
      CALL ERRCHK(IERR)
      CALL ADFCRE(PID,'DB1_NODE2',CID,IERR)
      CALL ERRCHK(IERR)
      CALL ADFSLB(CID,'LABEL IN FILE.1: NODE2',IERR)
      CALL ERRCHK(IERR)
      CALL ADFDCLO(RID,IERR)
      CALL ERRCHK(IERR)
C
C *** 1.) OPEN 2ND DATABASE
C
      CALL ADFDOPN('db2.adf','NEW',' ',RID,IERR)
      CALL ERRCHK(IERR)
      CALL ADFCRE(RID,'DB2_NODE1',PID,IERR)
      CALL ERRCHK(IERR)
      CALL ADFCRE(PID,'DB2_NODE2',CID,IERR)
      CALL ERRCHK(IERR)
C
C *** LINK NODE /DB1_NODE1/DB1_NODE2 TO /DB2_NODE1
C
      CALL ADFLINK(PID,'LINKED_NODE','db1.adf',
     X             '/DB1_NODE1/DB1_NODE2',CID,IERR)
      CALL ERRCHK(IERR)
C
C *** CHECK TO MAKE SURE THE NODE WAS ACTUALLY LINKED
C
      WRITE(*,160)
  160 FORMAT(/,'*** PARENT AFTER LINK ***')
      CALL PRTCLD(PID)
C
C *** FOR FINAL CONFIRMATION, GET ORIGINAL LABEL
C     GOING THROUGH NEW LINK
C
      CALL ADFGLB(CID,TSTLBL,IERR)
      CALL ERRCHK(IERR)
      WRITE(*,170)TSTLBL
  170 FORMAT(/,'LINKED_NODE LABEL = ',A)
C
      STOP
```

```
      END
C
C ************** SUBROUTINES ****************
C
      SUBROUTINE ERRCHK(IERR)
C
C *** CHECK ERROR CONDITION
C
      CHARACTER*80 MESS
      IF (IERR .GT. 0) THEN
         CALL ADFERR(IERR,MESS)
         PRINT *,MESS
         CALL ABORT('ADF ERROR')
      ENDIF
      RETURN
      END

      SUBROUTINE PRTCLD(PID)
C
C *** PRINT TABLE OF CHILDREN GIVEN A PARENT NODE-ID
C
      PARAMETER (MAXCLD=10)
      PARAMETER (MAXCHR=32)
      REAL*8 PID
      CHARACTER*(MAXCHR) NODNAM,NDNMS(MAXCLD)
      CALL ADFGNAM(PID,NODNAM,IERR)
      CALL ERRCHK(IERR)
      CALL ADFNCLD(PID,NUMC,IERR)
      CALL ERRCHK(IERR)
      WRITE(*,120)NODNAM,NUMC
  120 FORMAT(/,' PARENT NODE NAME = ',A,/,
     X       '     NUMBER OF CHILDREN = ',I2,/,
     X       '     CHILDREN NAMES:')
      NLEFT = NUMC
      ISTART = 1
C     --- TOP OF DO-WHILE LOOP
  130 CONTINUE
         CALL ADFCNAM(PID,ISTART,MAXCLD,LEN(NDNMS),
     X                NUMRET,NDNMS,IERR)
         CALL ERRCHK(IERR)
         WRITE(*,140)(NDNMS(K),K=1,NUMRET)
  140    FORMAT(2(8X,A))
         NLEFT = NLEFT - MAXCLD
         ISTART = ISTART + MAXCLD
      IF (NLEFT .GT. 0) GO TO 130
      RETURN
      END
```

The resulting output is:

```
 *** PARENT AFTER LINK ***

 PARENT NODE NAME = DB2_NODE1
     NUMBER OF CHILDREN =  2
     CHILDREN NAMES:
        DB2_NODE2                       LINKED_NODE

LINKED_NODE LABEL = LABEL IN FILE.1: NODE2
```

ADF_Is_Link — *See If the Node Is a Link*

ADF_Is_Link (ID,link_path_length,error_return)		
Language	C	Fortran
Routine Name	ADF_Is_Link	ADFISLK
Input	const double ID	real*8 ID
Output	int *link_path_length	integer link_path_length
	int *error_return	integer error_return

link_path_length　如果结点不是一个连接（link），那么返回值是0。

如果在同一个文件内，ID指向的是一个连接（link），路径的字符数将会被返回。如果指向的　是不同的文件的连接，指向文件的名称字符数与path+1的字符数之和将会被返回。

此子程序用来测试结点是否是一个连接（link）。如果实际的数据类型是LK，那么返回连接（link）的路径长度，否则为0。

示例：

```
      PROGRAM TEST
C
      PARAMETER (MAXCHR=32)
C
      CHARACTER*(MAXCHR) TSTLBL
C
C *** NODE IDS
C
      REAL*8 RID,PID,CID
      INTEGER IERR
C
C *** 1.) OPEN 1ST DATABASE
C     2.) CREATE TWO NODES
C     3.) PUT LABEL ON 2ND NODE
C     4.) CLOSE DATABASE
C
      CALL ADFDOPN('db1.adf','NEW',' ',RID,IERR)
      CALL ERRCHK(IERR)
      CALL ADFCRE(RID,'DB1_NODE1',PID,IERR)
      CALL ERRCHK(IERR)
      CALL ADFCRE(PID,'DB1_NODE2',CID,IERR)
```

```
      CALL ERRCHK(IERR)
      CALL ADFDCLO(RID,IERR)
      CALL ERRCHK(IERR)
C
C *** 1.) OPEN 2ND DATABASE
C
      CALL ADFDOPN('db2.adf','NEW',' ',RID,IERR)
      CALL ERRCHK(IERR)
      CALL ADFCRE(RID,'DB2_NODE1',PID,IERR)
      CALL ERRCHK(IERR)
      CALL ADFCRE(PID,'DB2_NODE2',CID,IERR)
      CALL ERRCHK(IERR)
C
C *** LINK NODE FILE 1:/DB1_NODE1/DB1_NODE2 TO /DB2_NODE1
C
      CALL ADFLINK(PID,'LINKED_NODE','db1.adf',
     X             '/DB1_NODE1/DB1_NODE2',CID,IERR)
      CALL ERRCHK(IERR)
C
C *** CHECK TO MAKE SURE THE NODE WAS ACTUALLY LINKED
C
      CALL ADFISLK(CID,LEN,IERR)
      CALL ERRCHK(IERR)
      PRINT *,' PATH LENGTH FROM LINK IS: ',LEN
C
      STOP
      END
C
C ************* SUBROUTINES ****************
C
      SUBROUTINE ERRCHK(IERR)
C
C *** CHECK ERROR CONDITION
C
      CHARACTER*80 MESS
      IF (IERR .GT. 0) THEN
         CALL ADFERR(IERR,MESS)
         PRINT *,MESS
         CALL ABORT('ADF ERROR')
      ENDIF
      RETURN
      END
```

The resulting output is:

```
 PATH LENGTH FROM LINK IS:           28
```

ADF_Get_Link_Path — *Get the Path Information From a Link*

ADF_Get_Link_Path (ID,file,name_in_file,error_return)

Language Routine Name	C ADF_Get_Link_Path	Fortran ADFGLKP
Input	const double ID	real*8 ID
Output	char *file char *name_in_file int *error_return	character*(*) file character*(*) name_in_file integer error_return

此子程序用来从一个连接中（link）提取路径信息。如果结点是一个连接，程序返回路径信息，否则返回错误。

示例：

```
      PROGRAM TEST
C
      PARAMETER (MAXCHR=32)
C
      CHARACTER*(MAXCHR) TSTLBL
      CHARACTER*(40) FILENM,PATH
C
C *** NODE IDS
C
      REAL*8 RID,PID,CID
      INTEGER IERR
C
C *** 1.) OPEN 1ST DATABASE
C     2.) CREATE TWO NODES
C     3.) PUT LABEL ON 2ND NODE
C     4.) CLOSE DATABASE
C
      CALL ADFDOPN('db1.adf','NEW',' ',RID,IERR)
      CALL ADFCRE(RID,'DB1_NODE1',PID,IERR)
```

```
      CALL ADFCRE(PID,'DB1_NODE2',CID,IERR)
      CALL ADFDCLO(RID,IERR)
C
C *** 1.) OPEN 2ND DATABASE
C
      CALL ADFDOPN('db2.adf','NEW',' ',RID,IERR)
      CALL ADFCRE(RID,'DB2_NODE1',PID,IERR)
      CALL ADFCRE(PID,'DB2_NODE2',CID,IERR)
C
C *** LINK NODE FILE 1:/DB1_NODE1/DB1_NODE2 TO /DB2_NODE1
C
      CALL ADFLINK(PID,'LINKED_NODE','db1.adf',
     X             '/DB1_NODE1/DB1_NODE2',CID,IERR)
C
C *** CHECK TO MAKE SURE THE NODE WAS ACTUALLY LINKED
C
      CALL ADFGLKP(CID,FILENM,PATH,IERR)
      PRINT *,' INFORMATION FROM LINK:'
      PRINT *,' FILE: ',FILENM
      PRINT *,' PATH: ',PATH
C
      STOP
      END
C
C ************* SUBROUTINES ****************
C
      SUBROUTINE ERRCHK(IERR)
C
C *** CHECK ERROR CONDITION
C
      CHARACTER*80 MESS
      IF (IERR .GT. 0) THEN
         CALL ADFERR(IERR,MESS)
         PRINT *,MESS
         CALL ABORT('ADF ERROR')
      ENDIF
      RETURN
      END

      SUBROUTINE PRTCLD(PID)
C
C *** PRINT TABLE OF CHILDREN GIVEN A PARENT NODE-ID
C
      PARAMETER (MAXCLD=10)
      PARAMETER (MAXCHR=32)
      REAL*8 PID
      CHARACTER*(MAXCHR) NODNAM,NDNMS(MAXCLD)
      CALL ADFGNAM(PID,NODNAM,IERR)
      CALL ERRCHK(IERR)
      CALL ADFNCLD(PID,NUMC,IERR)
      CALL ERRCHK(IERR)
```

```
      WRITE(*,120)NODNAM,NUMC
  120 FORMAT(/,' PARENT NODE NAME = ',A,/,
     X       '     NUMBER OF CHILDREN = ',I2,/,
     X       '     CHILDREN NAMES:')
      NLEFT = NUMC
      ISTART = 1
C     --- TOP OF DO-WHILE LOOP
  130 CONTINUE
         CALL ADFCNAM(PID,ISTART,MAXCLD,LEN(NDNMS),
     X                NUMRET,NDNMS,IERR)
         CALL ERRCHK(IERR)
         WRITE(*,140)(NDNMS(K),K=1,NUMRET)
  140    FORMAT(2(8X,A))
         NLEFT = NLEFT - MAXCLD
         ISTART = ISTART + MAXCLD
      IF (NLEFT .GT. 0) GO TO 130
      RETURN
      END
```

The resulting output is:

```
INFORMATION FROM LINK:
FILE: db1.adf
PATH: /DB1_NODE1/DB1_NODE2
```

ADF_Get_Root_ID — *Get the Root ID for the ADF System*

ADF_Get_Root_ID (ID,root_ID,error_return)		
Language	**C**	**Fortran**
Routine Name	ADF_Get_Root_ID	ADFGRID
Input	const double ID	real*8 ID
Output	double root_ID	real*8 root_ID
	int *error_return	integer error_return

当给定文件任何合法的结点ID，此子程序用来提取ADF文件的根结点ID。

示例：

```
PROGRAM TEST
C
      PARAMETER (MAXCHR=32)
C
      CHARACTER*(MAXCHR) NODNAM,ROOTNM
C
C *** NODE IDS
C
      REAL*8 RID,PID,CID,TESTID
      INTEGER I,J,IERR,NUMCLD
C
C *** OPEN DATABASE
C
      CALL ADFDOPN('db.adf','NEW',' ',RID,IERR)
      CALL ADFGNAM(RID,ROOTNM,IERR)
      PRINT *,' AFTER OPENING FILE, ROOT NAME = ',ROOTNM
C
C *** CREATE NODES AT FIRST LEVEL
C
      DO 150 I = 1,2
         WRITE(NODNAM,'(A7,I1)')'PARENT.',I
         CALL ADFCRE(RID,NODNAM,PID,IERR)
         TESTID = 0.0
         ROOTNM =''
         CALL ADFGRID(PID,TESTID,IERR)
```

```
            CALL ADFGNAM(TESTID,ROOTNM,IERR)
            WRITE(*,100)NODNAM,ROOTNM
  100       FORMAT('USING NODE ID FROM: ',A,
     X             ' ROOT NAME = ',A)
C
C ****** CREATE NODES AT SECOND LEVEL
C
            NUMCLD = 2*I
            DO 110 J = 1,NUMCLD
               WRITE(NODNAM,'(A6,I1,A1,I1)')'CHILD.',I,'.',J
               CALL ADFCRE(PID,NODNAM,CID,IERR)
               TESTID = 0.0
               ROOTNM =''
               CALL ADFGRID(PID,TESTID,IERR)
               CALL ADFGNAM(TESTID,ROOTNM,IERR)
               WRITE(*,100)NODNAM,ROOTNM
  110       CONTINUE
  150    CONTINUE
C
         STOP
         END
```

The resulting output is:

```
 AFTER OPENING FILE, ROOT NAME = ADF MotherNode
USING NODE ID FROM: PARENT.1          ROOT NAME = ADF MotherNode
USING NODE ID FROM: CHILD.1.1         ROOT NAME = ADF MotherNode
USING NODE ID FROM: CHILD.1.2         ROOT NAME = ADF MotherNode
USING NODE ID FROM: PARENT.2          ROOT NAME = ADF MotherNode
USING NODE ID FROM: CHILD.2.1         ROOT NAME = ADF MotherNode
USING NODE ID FROM: CHILD.2.2         ROOT NAME = ADF MotherNode
USING NODE ID FROM: CHILD.2.3         ROOT NAME = ADF MotherNode
USING NODE ID FROM: CHILD.2.4         ROOT NAME = ADF MotherNode
```

I. 5　数据查询子程序（data query）

ADF_Get_Label — *Get the String in a Node's Label Field*

ADF_Get_Label (ID,label,error_return)		
Language	C	Fortran
Routine Name	ADF_Get_Label	ADFGLB
Input	const double ID	real*8 ID
Output	char *label	character*(*) label
	int *error_return	integer error_return

此子程序用来返回存放在结点标签（Label）中的32-字符的字串。
示例：

```
PROGRAM TEST
C
      PARAMETER (MAXCHR=32)
C
      CHARACTER*(MAXCHR) NODNAM,LABL
C
C *** NODE IDS
C
      REAL*8 RID,CID
C
C *** OPEN DATABASE
C
      CALL ADFDOPN('db.adf','NEW',' ',RID,IERR)
      CALL ADFCRE(RID,'NODE 1',CID,IERR)
      CALL ADFSLB(CID,'THIS IS A NODE LABEL',IERR)
C
      CALL ADFGNAM(CID,NODNAM,IERR)
      CALL ADFGLB(CID,LABL,IERR)
C
      PRINT *,' NODE NAME = ',NODNAM
      PRINT *,' LABEL     = ',LABL
C
```

```
      STOP
      END
```

The resulting output is:

```
 NODE NAME = NODE 1
 LABEL     = THIS IS A NODE LABEL
```

ADF_Set_Label — *Set the String in a Node's Label Field*

ADF_Set_Label (ID,label,error_return)		
Language	C	Fortran
Routine Name	ADF_Set_Label	ADFSLB
Input	const double ID	real*8 ID
	const char *label	character*(*) label
Output	int *error_return	integer error_return

此子程序用来在结点的标签（Label）中设置32位的字串。
示例：见ADF_Get_Label中示例。

ADF_Get_Data_Type — *Get the String in a Node's Data-Type Field*

ADF_Get_Data_Type (ID,data_type,error_return)		
Language	C	Fortran
Routine Name	ADF_Get_Data_Type	ADFGDT
Input	const double ID	real*8 ID
Output	char *data_type	character*(*) data_type
	int *error_return	integer error_return

此子程序用来返回存放在结点数据类型区域中32位字串。
示例：

```
PROGRAM TEST
C
      PARAMETER (MAXCHR=32)
      PARAMETER (MAXROW=2)
      PARAMETER (MAXCOL=10)
C
      CHARACTER*(MAXCHR) NODNAM,LABL
      CHARACTER*(MAXCHR) DTYPE
      REAL R4DATI(MAXROW,MAXCOL),R4DATO(MAXROW,MAXCOL)
      INTEGER IDIMI(2),IDIMO(2)
C
C *** NODE IDS
C
      REAL*8 RID,CID
C
C *** OPEN DATABASE
C
      CALL ADFDOPN('db.adf','NEW',' ',RID,IERR)
C
C *** GENERATE SOME DATA
C
      IDIMI(1) = MAXROW
      IDIMI(2) = MAXCOL
```

```
      DO 200 ICOL = 1,MAXCOL
         DO 100 IROW = 1,MAXROW
            R4DATI(IROW,ICOL) = 2.0*ICOL*IROW
  100    CONTINUE
  200 CONTINUE
C
C *** GENERATE A NODE AND PUT DATA IN IT
C
      CALL ADFCRE(RID,'NODE 1',CID,IERR)
      CALL ADFSLB(CID,'LABEL FOR NODE 1',IERR)
      CALL ADFPDIM(CID,'R4',2,IDIMI,IERR)
      CALL ADFWALL(CID,R4DATI,IERR)
C
C *** GET INFORMATION FROM NODE
C
      CALL ADFGNAM(CID,NODNAM,IERR)
      CALL ADFGLB(CID,LABL,IERR)
      CALL ADFGDT(CID,DTYPE,IERR)
      CALL ADFGND(CID,NDIM,IERR)
      CALL ADFGDV(CID,IDIMO,IERR)
      CALL ADFRALL(CID,R4DATO,IERR)
C
      PRINT *,' NODE NAME            = ',NODNAM
      PRINT *,' LABEL                = ',LABL
      PRINT *,' DATA TYPE            = ',DTYPE
      PRINT *,' NUMBER OF DIMENSIONS = ',NDIM
      PRINT *,' DIMENSIONS           = ',IDIMO
      PRINT *,' DATA:'
      WRITE(*,300)((R4DATO(I,J),I=1,MAXROW),J=1,MAXCOL)
  300 FORMAT(2(5X,F10.2))
C
      STOP
      END
```

The resulting output is:

```
 NODE NAME            = NODE 1
 LABEL                = LABEL FOR NODE 1
 DATA TYPE            = R4
 NUMBER OF DIMENSIONS =            2
 DIMENSIONS           =            2           10
 DATA:
          2.00           4.00
          4.00           8.00
          6.00          12.00
          8.00          16.00
         10.00          20.00
         12.00          24.00
         14.00          28.00
         16.00          32.00
         18.00          36.00


 20.00          40.00
```

ADF_Get_Number_of_Dimensions — *Get the Number of Node Dimensions*

ADF_Get_Number_of_Dimensions (ID,num_dims,error_return)		
Language	**C**	**Fortran**
Routine Name	ADF_Get_Number_of_Dimensions	ADFGND
Input	const double ID	real*8 ID
Output	int *num_dims	integer num_dims
	int *error_return	integer error_return

此子程序用来返回结点中数据的维度值。返回的值只是结点中定义的维度。如果维度值为0，返回错误。

示例：见ADF_Get_Data_Type中示例。

ADF_Get_Dimension_Values — *Get the Values of the Node Dimensions*

ADF_Get_Dimension_Values (ID,dim_vals[],error_return)		
Language	**C**	**Fortran**
Routine Name	ADF_Get_Dimension_Values	ADFGDV
Input	const double ID	real*8 ID
Output	int *dim_vals[]	integer dim_vals()
	int *error_return	integer error_return

此子程序返回结点的维度的数组列表值。

示例：见ADF_Get_Data_Type中的示例。

ADF_Put_Dimension_Information — *Set or Change the Data Type and Dimensions of a Node*

ADF_Put_Dimension_Information (ID,data_type,dims,dim_vals[],error_return)		
Language	**C**	**Fortran**
Routine Name	ADF_Put_Dimension_Information	ADFPDIM
Input	const double ID	real*8 ID
	const char *data_type	character*(*) data_type
	const int dims	integer dims
	int dim_vals[]	integer dim_vals()
Output	int *error_return	integer error_return

dims　　结点的维度值。取值为0到12。0意味着没有数据。

此子程序用来设置或更改结点的类型数据和维度信息。

注意：当此程序用来设置或更改结点数据类型和维度信息时，与该结点相关的所有数据都会丢失。可根据需要，调整数据的维度和扩展数据的空间。改变维度时要格外的小心。数据的层次结构与Fortran相同。

示例：见ADF_Get_Data_Type中的示例。

I. 6 数据的 I/O 子程序

注意：对所有的数据I/O,系统对索引的起始点都是1而非0，就是索引数组中第一个元素序号是1而非0。

ADF_Read_Data — *Read the Data From a Node Having Stride Capabilities*

ADF_Read_Data (ID,s_start[],s_end[],s_stride[],m_num_dims,m_dims[],m_start[], m_end[],m_stride[],data,error_return)

Language	C	Fortran
Routine Name	ADF_Read_Data	ADFREAD
Input	const double ID	real*8 ID
	const int s_start[]	integer s_start()
	const int s_end[]	integer s_end()
	const int s_stride[]	integer s_stride()
	const int m_num_dims	integer m_num_dims
	const int m_dims[]	integer m_dims()
	const int m_start[]	integer m_start()
	const int m_end[]	integer m_end()
	const int m_stride[]	integer m_stride()
Output	char *data	character*(*) data
	int *error_return	integer error_return

s_start[]　用来索引数据库中结点数组的每一个维度的索引表的起始位，ADF中允许的最大的维度值为12。

s_end[]　用来索引数据库中结点数组的每一个维度的索引表的终止位。

s_stride[]　用来索引数据库中结点数组的每一个维度的步长值。

m_dims[]　内存中数组的维度值。

m_start[]　内存中数组每一维度的起始位。

m_end[]　内存中数组每一维度的终止位。

m_stride[]　内存中数组每一维度的步长值。

此子程序用来提供一般目的上的数据读取功能。提供通用的数据内的初始位置，以及从数据初始位置的步长跳跃。这个功能对磁盘和内存中数据同样有效。一组整数矢量用来描述结点内的数据分布，另外一组矢量用来描述内存中相应的数据分布。

使用ADF_Read_Data与相比在效率上会有明显的损失。如果效率优先的话，最好把数据进行组织来利用ADF_Read_All_Data的效率优势。

数据在内存和磁盘都有存储，意味着第一索引变化是最快的。

是不会接受“negative”的索引。意味着把结点的数据传到内存中时数据顺序打乱是不可能的。

当用ADF_Write_All_Data写数据和用ADF_Read_Data随机读取数据要相当的小心。

注意：如果结点的所有数据类型是复合型数据类型，如“I4[3]，R8”,部分的功能会存取一个或更多的20-byte的数据体。你不能存取其子集。

示例：

```
PROGRAM TEST
C
      PARAMETER (MAXROW=10)
      PARAMETER (MAXCOL=3)
C
      REAL R4ARRI(MAXROW,MAXCOL)
      REAL R4ARRO(MAXROW,MAXCOL)
      INTEGER IDIMI(2),IDIMO(2)
      INTEGER IDBEG(2),IDEND(2),IDINCR(2)
      INTEGER IMBEG(2),IMEND(2),IMINCR(2)
C
C *** NODE IDS
C
      REAL*8 RID,CID
C
```

```
C *** OPEN DATABASE
C
      CALL ADFDOPN('db.adf','NEW',' ',RID,IERR)
C
C *** GENERATE SOME DATA
C
      IDIMI(1) = MAXROW
      IDIMI(2) = MAXCOL
      DO 200 ICOL = 1,MAXCOL
         DO 100 IROW = 1,MAXROW
            R4ARRI(IROW,ICOL) = 2.0*ICOL*IROW
  100    CONTINUE
  200 CONTINUE
      PRINT *,' ORIGINAL ARRAY:'
      WRITE(*,300)((R4ARRI(I,J),J=1,MAXCOL),I=1,MAXROW)
  300 FORMAT(3(5X,F10.2))
C
C *** GENERATE A NODE AND PUT DATA IN IT
C
      CALL ADFCRE(RID,'NODE 1',CID,IERR)
      CALL ADFSLB(CID,'LABEL FOR NODE 1',IERR)
      CALL ADFPDIM(CID,'R4',2,IDIMI,IERR)
      CALL ADFWALL(CID,R4ARRI,IERR)
C
C *** GET INFORMATION FROM NODE
C
C *** GET DATA FROM NODE (EXACTLY EQUIVALENT TO ADFRALL)
C
      IDBEG(1) = 1
      IDEND(1) = MAXROW
      IDINCR(1) = 1
C
      IDBEG(2) = 1
      IDEND(2) = MAXCOL
      IDINCR(2) = 1
C
      IDIMO(1) = MAXROW
      IDIMO(2) = MAXCOL
C
      IMBEG(1) = 1
      IMEND(1) = MAXROW
      IMINCR(1) = 1
C
      IMBEG(2) = 1
      IMEND(2) = MAXCOL
      IMINCR(2) = 1
      CALL ADFREAD(CID,IDBEG,IDEND,IDINCR,
     X             2,IDIMO,IMBEG,IMEND,IMINCR,
     X             R4ARRO,IERR)
      CALL ERRCHK(IERR)
C
```

```
      PRINT *,' ARRAY PULLED FROM DISK USING ADFREAD:'
      WRITE(*,300)((R4ARR0(I,J),J=1,MAXCOL),I=1,MAXROW)
C
      STOP
      END

C
C ************* SUBROUTINES ****************
C
      SUBROUTINE ERRCHK(IERR)
C
C *** CHECK ERROR CONDITION
C
      CHARACTER*80 MESS
      IF (IERR .GT. 0) THEN
         CALL ADFERR(IERR,MESS)
         PRINT *,MESS
         CALL ABORT('ADF ERROR')
      ENDIF
      RETURN
      END
```

The resulting output is:

```
 ORIGINAL ARRAY:
           2.00           4.00           6.00
           4.00           8.00          12.00
           6.00          12.00          18.00
           8.00          16.00          24.00
          10.00          20.00          30.00
          12.00          24.00          36.00
          14.00          28.00          42.00
          16.00          32.00          48.00
          18.00          36.00          54.00
          20.00          40.00          60.00

 ARRAY PULLED FROM DISK USING ADFREAD:
           2.00           4.00           6.00
           4.00           8.00          12.00
           6.00          12.00          18.00
           8.00          16.00          24.00
          10.00          20.00          30.00
          12.00          24.00          36.00
          14.00          28.00          42.00
          16.00          32.00          48.00
          18.00          36.00          54.00
          20.00          40.00          60.00
```

示例2：

```
      PROGRAM TEST
C
      PARAMETER (MAXROW=10)
      PARAMETER (MAXCOL=3)
C
      REAL R4ARRI(MAXROW,MAXCOL),R4VECO(MAXROW)
      INTEGER IDIMD(2)
      INTEGER IDBEG(2),IDEND(2),IDINCR(2)
C
C *** NODE IDS
C
      REAL*8 RID,CID
C
C *** OPEN DATABASE
C
      CALL ADFDOPN('db.adf','NEW',' ',RID,IERR)
C
C *** GENERATE SOME DATA
C
      IDIMD(1) = MAXROW
      IDIMD(2) = MAXCOL
      DO 200 ICOL = 1,MAXCOL
         DO 100 IROW = 1,MAXROW
            R4ARRI(IROW,ICOL) = 2.0*ICOL*IROW
  100    CONTINUE
  200 CONTINUE
C
      DO 250 I = 1,MAXROW
         R4VECO(I) = 0.0
  250 CONTINUE
C
      PRINT *,' ORIGINAL ARRAY:'
      WRITE(*,300)((R4ARRI(I,J),J=1,MAXCOL),I=1,MAXROW)
  300 FORMAT(3(5X,F10.2))
C
C *** GENERATE A NODE AND PUT DATA IN IT
C
      CALL ADFCRE(RID,'NODE 1',CID,IERR)
      CALL ADFSLB(CID,'LABEL FOR NODE 1',IERR)
      CALL ADFPDIM(CID,'R4',2,IDIMD,IERR)
      CALL ADFWALL(CID,R4ARRI,IERR)
C
C *** GET DATA FROM NODE USING STRIDED READ
C
C ****** TAKE EVERY OTHER NUMBER FROM THE 2ND COLUMN OF THE ARRAY
C        AND PUT IT IN SEQUENTIALLY IN A VECTOR IN MEMORY
C
C *** DATABASE STRIDE INFORMATION
```

```
C
      IDBEG(1) = 1
      IDEND(1) = MAXROW
      IDINCR(1) = 2
C
      IDBEG(2) = 2
      IDEND(2) = 2
      IDINCR(2) = 1
C
C *** MEMORY STRIDE INFORMATION
C
      NDIMM = 1
      IDIMM = MAXROW
      IMBEG = 1
      IMEND = MAXROW
      IMINCR = 2
C
      CALL ADFREAD(CID,IDBEG,IDEND,IDINCR,
     X             NDIMM,IDIMM,IMBEG,IMEND,IMINCR,
     X             R4VECO,IERR)
      CALL ERRCHK(IERR)
C
      PRINT *,' VECTOR WITH DATA EXTRACTED FROM ARRAY'
      WRITE(*,400)(R4VECO(J),J=1,MAXROW)
  400 FORMAT(3(5X,F10.2))
C
      STOP
      END

C
C ************* SUBROUTINES ****************
C
      SUBROUTINE ERRCHK(IERR)
C
C *** CHECK ERROR CONDITION
C
      CHARACTER*80 MESS
      IF (IERR .GT. 0) THEN
         CALL ADFERR(IERR,MESS)
         PRINT *,MESS
         CALL ABORT('ADF ERROR')
      ENDIF
      RETURN
      END
```

The resulting output is:

```
 ORIGINAL ARRAY:
          2.00          4.00          6.00
          4.00          8.00         12.00
          6.00         12.00         18.00
```

```
            8.00          16.00          24.00
           10.00          20.00          30.00
           12.00          24.00          36.00
           14.00          28.00          42.00
           16.00          32.00          48.00
           18.00          36.00          54.00
           20.00          40.00          60.00
 VECTOR WITH DATA EXTRACTED FROM ARRAY
            4.00           0.00          12.00
            0.00          20.00           0.00
           28.00           0.00          36.00
            0.00
```

ADF_Read_All_Data — *Read All the Data From a Node*

ADF_Read_All_Data (ID,data,error_return)		
Language	**C**	**Fortran**
Routine Name	ADF_Read_All_Data	ADFRALL
Input	const double ID	real*8 ID
Output	char *data	character*(*) data
	int *error_return	integer error_return

此子程序用来从一个结点读取所有的数据。它读取所有的结点数据并返回数据到连续相邻的内存空间中。

示例：见ADF_Get_Data_Type中的示例。

ADF_Read_Block_Data — *Read a Contiguous Block of Data From a Node*

ADF_Read_Block_Data (ID,b_start,b_end,data,error_return)		
Language	**C**	**Fortran**
Routine Name	ADF_Read_Block_Data	ADFRBLK
Input	const double ID	real*8 ID
	const long b_start	integer b_start
	const long b_end	integer b_end
Output	char *data	character*(*) data
	int *error_return	integer error_return

b_start　　空间内数据块的起始点

b_end　　空间内数据块的终止点。

此子程序用来读取结点的一个数据块的数据，并返回到连续相邻的内存中。

ADF_Write_Data — *Write the Data to a Node Having Stride Capabilities*

ADF_Write_Data (ID,s_start[],s_end[],s_stride[],m_num_dims,m_dims[],m_start[], m_end[],m_stride[],data,error_return)		
Language	C	Fortran
Routine Name	ADF_Write_Data	ADFWRIT
Input	const double ID const int s_start[] const int s_end[] const int s_stride[] const int m_num_dims const int m_dims[] const int m_start[] const int m_end[] const int m_stride[] char *data	real*8 ID integer s_start() integer s_end() integer s_stride() integer m_num_dims integer m_dims() integer m_start() integer m_end() integer m_stride() character*(*) data
Output	int *error_return	integer error_return

此子程序用来提供一般目的的数据写入功能。
示例1：

```
      PROGRAM TEST
C
      PARAMETER (MAXCHR=32)
      PARAMETER (MAXROW=10)
      PARAMETER (MAXCOL=3)
C
      CHARACTER*(MAXCHR) NODNAM,LABL
      CHARACTER*(MAXCHR) DTYPE
      REAL R4ARRI(MAXROW,MAXCOL)
      REAL R4ARRO(MAXROW,MAXCOL)
      INTEGER IDIMI(2),IDIMO(2)
      INTEGER IDBEG(2),IDEND(2),IDINCR(2)
      INTEGER IMBEG(2),IMEND(2),IMINCR(2)
C
C *** NODE IDS
C
      REAL*8 RID,CID
C
C *** OPEN DATABASE
C
      CALL ADFDOPN('db.adf','NEW',' ',RID,IERR)
C
C *** GENERATE SOME DATA
C
      IDIMI(1) = MAXROW
      IDIMI(2) = MAXCOL
```

```
      DO 200 ICOL = 1,MAXCOL
         DO 100 IROW = 1,MAXROW
            R4ARRI(IROW,ICOL) = 2.0*ICOL*IROW
  100    CONTINUE
  200 CONTINUE
      PRINT *,' ORIGINAL ARRAY:'
      WRITE(*,300)((R4ARRI(I,J),J=1,MAXCOL),I=1,MAXROW)
  300 FORMAT(3(5X,F10.2))
C
C *** GENERATE A NODE AND PUT DATA IN IT
C     THIS IS EXACTLY EQUIVALENT TO USING ADFWALL
C
      CALL ADFCRE(RID,'NODE 1',CID,IERR)
      CALL ADFSLB(CID,'LABEL FOR NODE 1',IERR)
      CALL ADFPDIM(CID,'R4',2,IDIMI,IERR)
C
      IDBEG(1) = 1
      IDEND(1) = MAXROW
      IDINCR(1) = 1
C
      IDBEG(2) = 1
      IDEND(2) = MAXCOL
      IDINCR(2) = 1
C
      IDIMO(1) = MAXROW
      IDIMO(2) = MAXCOL
C
      IMBEG(1) = 1
      IMEND(1) = MAXROW
      IMINCR(1) = 1
C
      IMBEG(2) = 1
      IMEND(2) = MAXCOL
      IMINCR(2) = 1
C
      CALL ADFWRIT(CID,IDBEG,IDEND,IDINCR,2,IDIMO,IMBEG,
     X             IMEND,IMINCR,R4ARRI,IERR)
      CALL ERRCHK(IERR)
C
C *** GET INFORMATION FROM NODE
C
      CALL ADFGNAM(CID,NODNAM,IERR)
      CALL ADFGLB(CID,LABL,IERR)
      CALL ADFGDT(CID,DTYPE,IERR)
      CALL ADFGND(CID,NDIM,IERR)
      CALL ADFGDV(CID,IDIMO,IERR)
      CALL ADFRALL(CID,R4ARRO,IERR)
      CALL ERRCHK(IERR)
C
      PRINT *,' '
      PRINT *,' NODE NAME            = ',NODNAM
```

```
      PRINT *,' LABEL                = ',LABL
      PRINT *,' DATA TYPE            = ',DTYPE
      PRINT *,' NUMBER OF DIMENSIONS = ',NDIM
      PRINT *,' DIMENSIONS           = ',IDIMO
      PRINT *,' ADFRALL DATA:'
      WRITE(*,300)((R4ARRO(I,J),J=1,MAXCOL),I=1,MAXROW)
C
      STOP
      END

C
C ************* SUBROUTINES ****************
C
      SUBROUTINE ERRCHK(IERR)
C
C *** CHECK ERROR CONDITION
C
      CHARACTER*80 MESS
      IF (IERR .GT. 0) THEN
         CALL ADFERR(IERR,MESS)
         PRINT *,MESS
         CALL ABORT('ADF ERROR')
      ENDIF
      RETURN
      END
```

The resulting output is:

```
ORIGINAL ARRAY:
          2.00          4.00          6.00
          4.00          8.00         12.00
          6.00         12.00         18.00
          8.00         16.00         24.00
         10.00         20.00         30.00
         12.00         24.00         36.00
         14.00         28.00         42.00
         16.00         32.00         48.00
         18.00         36.00         54.00
         20.00         40.00         60.00

NODE NAME            = NODE 1
LABEL                = LABEL FOR NODE 1
DATA TYPE            = R4
NUMBER OF DIMENSIONS =           2
DIMENSIONS           =          10           3
ADFRALL DATA:
          2.00          4.00          6.00
          4.00          8.00         12.00
          6.00         12.00         18.00
          8.00         16.00         24.00
         10.00         20.00         30.00
```

```
     12.00          24.00          36.00
     14.00          28.00          42.00
     16.00          32.00          48.00
     18.00          36.00          54.00
     20.00          40.00          60.00
```

示例2：

```
      PROGRAM TEST
      C
            PARAMETER (MAXCHR=32)
            PARAMETER (MAXROW=10)
            PARAMETER (MAXCOL=3)
      C
            CHARACTER*(MAXCHR) NODNAM,LABL
            CHARACTER*(MAXCHR) DTYPE
            REAL R4ARRI(MAXROW,MAXCOL),R4VEC(MAXCOL)
            REAL R4ARRO(MAXROW,MAXCOL)
            INTEGER IDIMI(2),IDIMO(2),IDIMM(2)
            INTEGER IDBEG(2),IDEND(2),IDINCR(2)
            INTEGER IMBEG(2),IMEND(2),IMINCR(2)
      C
      C *** NODE IDS
      C
            REAL*8 RID,CID
      C
      C *** OPEN DATABASE
      C
            CALL ADFDOPN('db.adf','NEW',' ',RID,IERR)
      C
      C *** GENERATE SOME DATA
      C
            IDIMI(1) = MAXROW
            IDIMI(2) = MAXCOL
            DO 200 ICOL = 1,MAXCOL
               DO 100 IROW = 1,MAXROW
                  R4ARRI(IROW,ICOL) = 2.0*ICOL*IROW
        100    CONTINUE
               R4VEC(ICOL) = 2.2*ICOL
        200 CONTINUE
            PRINT *,' ORIGINAL ARRAY:'
            WRITE(*,300)((R4ARRI(I,J),J=1,MAXCOL),I=1,MAXROW)
        300 FORMAT(3(5X,F10.2))
      C
      C *** GENERATE A NODE AND WRITE THE ARRAY IN IT
      C
            CALL ADFCRE(RID,'NODE 1',CID,IERR)
```

```
      CALL ADFSLB(CID,'LABEL FOR NODE 1',IERR)
      CALL ADFPDIM(CID,'R4',2,IDIMI,IERR)
      CALL ADFWALL(CID,R4ARRI,IERR)
      CALL ERRCHK(IERR)
C
C *** GET INFORMATION FROM NODE (JUST TO PROVE ITS RIGHT)
C
      CALL ADFGNAM(CID,NODNAM,IERR)
      CALL ADFGLB(CID,LABL,IERR)
      CALL ADFGDT(CID,DTYPE,IERR)
      CALL ADFGND(CID,NDIM,IERR)
      CALL ADFGDV(CID,IDIMO,IERR)
      CALL ADFRALL(CID,R4ARRO,IERR)
      CALL ERRCHK(IERR)
C
      PRINT *,' '
      PRINT *,' NODE NAME            = ',NODNAM
      PRINT *,' LABEL                = ',LABL
      PRINT *,' DATA TYPE            = ',DTYPE
      PRINT *,' NUMBER OF DIMENSIONS = ',NDIM
      PRINT *,' DIMENSIONS           = ',IDIMO
      PRINT *,' ORIGINAL DATA ON DISK:'
      WRITE(*,300)((R4ARRO(I,J),J=1,MAXCOL),I=1,MAXROW)
C
C *** NOW, USING A VECTOR WITH NEW DATA IN IT, SCATTER
C     IT INTO THE DATABASE (THIS MODIFIES THE 5TH ROW
C     OF THE MATRIX)
C
      IDBEG(1)  = 5
      IDEND(1)  = 5
      IDINCR(1) = 1
C
      IDBEG(2)  = 1
      IDEND(2)  = MAXCOL
      IDINCR(2) = 1
C
      NMDIM = 1
      IDIMM(1)  = MAXCOL
      IMBEG(1)  = 1
      IMEND(1)  = MAXCOL
      IMINCR(1) = 1
C
      CALL ADFWRIT(CID,IDBEG,IDEND,IDINCR,
     X             NMDIM,IDIMM,IMBEG,IMEND,IMINCR,
     X             R4VEC,IERR)
      CALL ERRCHK(IERR)
C
C *** NOW PULL THE REVISED ARRAY OFF DISK AND PRINT IT
C
      CALL ADFRALL(CID,R4ARRO,IERR)
      CALL ERRCHK(IERR)
```

```
C
      PRINT *,' '
      PRINT *,' AFTER SCATTER:'
      WRITE(*,300)((R4ARR0(I,J),J=1,MAXCOL),I=1,MAXROW)
C
STOP
END

C
C ************* SUBROUTINES ****************
C
      SUBROUTINE ERRCHK(IERR)
C
C *** CHECK ERROR CONDITION
C
      CHARACTER*80 MESS
      IF (IERR .GT. 0) THEN
         CALL ADFERR(IERR,MESS)
         PRINT *,MESS
         CALL ABORT('ADF ERROR')
      ENDIF
      RETURN
      END
```

The resulting output is:

```
ORIGINAL ARRAY:
          2.00           4.00           6.00
          4.00           8.00          12.00
          6.00          12.00          18.00
          8.00          16.00          24.00
         10.00          20.00          30.00
         12.00          24.00          36.00
         14.00          28.00          42.00
         16.00          32.00          48.00
         18.00          36.00          54.00
         20.00          40.00          60.00

NODE NAME            = NODE 1
LABEL                = LABEL FOR NODE 1
DATA TYPE            = R4
NUMBER OF DIMENSIONS =           2
DIMENSIONS           =          10           3
ORIGINAL DATA ON DISK:
          2.00           4.00           6.00
          4.00           8.00          12.00
          6.00          12.00          18.00
          8.00          16.00          24.00
         10.00          20.00          30.00
         12.00          24.00          36.00
         14.00          28.00          42.00
```

```
          16.00          32.00          48.00
          18.00          36.00          54.00
          20.00          40.00          60.00
 AFTER SCATTER:
           2.00           4.00           6.00
           4.00           8.00          12.00
           6.00          12.00          18.00
           8.00          16.00          24.00
           2.20           4.40           6.60
          12.00          24.00          36.00
          14.00          28.00          42.00
          16.00          32.00          48.00
          18.00          36.00          54.00
          20.00          40.00          60.00
```

ADF_Write_All_Data — *Write All the Data to a Node*

ADF_Write_All_Data (ID,data,error_return)		
Language	C	Fortran
Routine Name	ADF_Write_All_Data	ADFWALL
Input	const double ID	real*8 ID
	const char *data	character*(*) data
Output	int *error_return	integer error_return

此子程序用来把所有数据写入结点。它把所有数据从连续和相邻的内存空间拷贝到连续和相邻的磁盘空间。

示例：见ADF_Get_Data_Type中的示例。

ADF_Write_Block_Data — *Write a Contiguous Block of Data To a Node*

ADF_Write_Block_Data (ID,b_start,b_end,data,error_return)		
Language	C	Fortran
Routine Name	ADF_Write_Block_Data	ADFWBLK
Input	const double ID	real*8 ID
	const long b_start	integer b_start
	const long b_end	integer b_end
	char *data	character*(*) data
Output	int *error_return	integer error_return

此子程序用来把连续内存空间中的数据块写入结点。

I.7 其它程序介绍

ADF_Flush_to_Disk — *Flush the Data to the Disk*

ADF_Flush_to_Disk (ID,error_return)		
Language	C	Fortran
Routine Name	ADF_Flush_to_Disk	ADFFTD
Input	const double ID	real*8 ID
Output	int *error_return	integer error_return

ADF_Database_Garbage_Collection — *Flush the Data to the Disk*

ADF_Database_Garbage_Collection (ID,error_return)		
Language	C	Fortran
Routine Name	ADF_Database_Garbage_Collection	ADFDGC
Input	const double ID	real*8 ID
Output	int *error_return	integer error_return

ADF_Error_Message — *Get a Description of the Error*

ADF_Error_Message (error_code,error_string)		
Language	C	Fortran
Routine Name	ADF_Error_Message	ADFERR
Input	const int error_code	integer error_code
Output	char *error_string	character*(*) error_string

示例：见ADF_Set_Error_State。

ADF_Set_Error_State — *Set the Error State Flag*

ADF_Set_Error_State (error_state,error_return)		
Language	C	Fortran
Routine Name	ADF_Set_Error_State	ADFSES
Input	const int error_state	integer error_state
Output	int *error_return	integer error_return

示例：

```
PROGRAM TEST
C
      PARAMETER (MAXCHR=32)
C
      CHARACTER*(MAXCHR) NODNAM
      CHARACTER*(80) MESS
C
C *** NODE IDS
C
      REAL*8 RID,CID1,CID2
C
C *** OPEN DATABASE
C
      CALL ADFDOPN('db.adf','NEW',' ',RID,IERR)
C
C *** CREATE 2 NODES
C
      CALL ADFCRE(RID,'NODE 1',CID1,IERR)
      CALL ADFCRE(RID,'NODE 2',CID2,IERR)
C
      CALL ADFGES(IDEFS,IERR)
      PRINT *,' DEFAULT ERROR STATE = ',IDEFS
C
C *** REQUEST NODE NAME FOR A NODE THAT DOES NOT EXIST
C
      PRINT *,' *** ON ERROR CONTINUE'
```

```
      CALL ADFGNAM(CID3,NODNAM,IERR)
      CALL ADFERR(IERR,MESS)
      PRINT *,'     ADF ERROR OCURRED, MESSAGE: ',MESS
      PRINT *,' '
C
C *** SET ABORT ON ERROR FLAG
C
      INEWS = 1
      CALL ADFSES(INEWS,IERR)
      PRINT *,' *** ABORT ON ERROR SET'
C
C *** REQUEST NODE NAME FOR A NODE THAT DOES NOT EXIST
C
      CALL ADFGNAM(CID3,NODNAM,IERR)
      PRINT *,' HELLO WORLD'
C
      STOP
      END
```

The resulting output is:

```
DEFAULT ERROR STATE =            0
  *** ON ERROR CONTINUE
      ADF ERROR OCURRED, MESSAGE:
ADF 10: ADF file index out of legal range.

  *** ABORT ON ERROR SET
ADF 10: ADF file index out of legal range.
ADF Aborted: Exiting
```

ADF_Get_Error_State — *Get the Error State*

ADF_Get_Error_State (error_state,error_return)		
Language	**C**	**Fortran**
Routine Name	ADF_Get_Error_State	ADFGES
Input		
Output	int *error_state	integer error_state
	int *error_return	integer error_return

示例：

```
PROGRAM TEST
C
C *** NODE IDS
C
      REAL*8 RID
C
C *** OPEN DATABASE
C
      CALL ADFDOPN('db.adf','NEW',' ',RID,IERR)
C
C *** CHECK THE DEFAULT ERROR STATE
C
      CALL ADFGES(IDEFS,IERR)
C
C *** SET THE ERROR STATE TO ABORT ON ERROR
C
      CALL ADFSES(1,IERR)
C
C *** MAKE SURE STATE WAS SET AS DESIRED
C
      CALL ADFGES(NDEFS,IERR)
C
C *** PRINT OUT RESULTS
C
      PRINT *,' DEFAULT ERROR STATE = ',IDEFS
      PRINT *,' RESET ERROR STATE   = ',NDEFS
C
      STOP
      END
```

The resulting output is:

```
 DEFAULT ERROR STATE =            0
 RESET ERROR STATE   =            1
```

ADF_Database_Version — *Get the Version Number of the ADF Library That Created the ADF Database*

```
ADF_Database_Version (root_ID,version,creation_date,modification_date,
                      error_return)
```

Language	C	Fortran
Routine Name	ADF_Database_Version	ADFDVER
Input	constant double root_ID	real*8 root_ID
Output	char *version	character*(*) version
	char *creation_date	character*(*) creation_date
	char *modification_date	character*(*) modification_date
	int *error_return	integer error_return

示例：

```
      PROGRAM TEST
C
      PARAMETER (MAXCHR=32)
C
      CHARACTER*(MAXCHR) NODNAM
      CHARACTER*(MAXCHR) CVER,LVER,CDATE,MDATE
      CHARACTER*(80) MESS
C
C *** NODE IDS
C
      REAL*8 RID,CID1,CID2
C
C *** OPEN DATABASE
C
      CALL ADFDOPN('db.adf','NEW',' ',RID,IERR)
```

```
C
C *** CREATE 2 NODES
C
      CALL ADFCRE(RID,'NODE 1',CID1,IERR)
      CALL ADFCRE(RID,'NODE 2',CID2,IERR)
C
      CALL ADFDVER(RID,CVER,CDATE,MDATE,IERR)
      CALL ADFLVER(LVER,IERR)
      PRINT *,' VERSION INFORMATION:'
      PRINT *,'    ADF LIBRARY USED FOR CREATION: ',CVER
      PRINT *,'    CREATION DATE                : ',CDATE
      PRINT *,'    MODIFICATION DATE            : ',MDATE
      PRINT *,'    ADF LIBRARY BEING USED       : ',LVER
C
      STOP
      END
```

The resulting output is:

```
 VERSION INFORMATION:
    ADF LIBRARY USED FOR CREATION: ADF Database Version A01007
    CREATION DATE                : Thu Apr 24 15:41:55 1997
    MODIFICATION DATE            : Thu Apr 24 15:41:55 1997
    ADF LIBRARY BEING USED       : ADF Library  Version C01
```

ADF_Library_Version — *Get the Version Number of the ADF Library That the Application Program is Currently Using*

ADF_Library_Version (version,error_return)

Language	C	Fortran
Routine Name	ADF_Library_Version	ADFLVER
Input		
Output	char *version int *error_return	character*(*) version integer error_return

version　一个32位的字符串包含了ADF库的版本ID信息。

示例：见ADF_Database_Version。

J　Fortran 程序样本

下面的程序样本建立的ADF文件见图1。

```
      PROGRAM TEST
C
C     SAMPLE ADF TEST PROGRAM TO BUILD ADF FILES ILLUSTRATED
C     IN THE EXAMPLE DATABASE FIGURE
C
      PARAMETER (MAXCHR=32)
C
      CHARACTER*(MAXCHR) TSTLBL,DTYPE
      CHARACTER*(MAXCHR) FNAM,PATH
C
      REAL*8 RID,PID,CID,TMPID,RIDF2
      REAL A(4,3),B(4,3)
      INTEGER IC(6),ID(6)
      INTEGER IERR
      INTEGER IDIM(2),IDIMA(2),IDIMC,IDIMD
C
      DATA A /1.1,2.1,3.1,4.1,
     X        1.2,2.2,3.2,4.2,
     X        1.3,2.3,3.3,4.3/
      DATA IDIMA /4,3/
C
      DATA IC /1,2,3,4,5,6/
      DATA IDIMC /6/
C
C     SET ERROR FLAG TO ABORT ON ERROR
C
      CALL ADFSES(1,IERR)
C
C *** 1.) OPEN 1ST DATABASE (ADF_FILE_TWO.ADF)
C     2.) CREATE THREE NODES AT FIRST LEVEL
C     3.) PUT LABEL ON NODE F3
C     4.) PUT DATA IN F3
C     5.) CREATE TWO NODES BELOW F3
C     6.) CLOSE DATABASE
C
      CALL ADFDOPN('adf_file_two.adf','NEW',' ',RID,IERR)
      RIDF2 = RID
      CALL ADFCRE(RID,'F1',TMPID,IERR)
      CALL ADFCRE(RID,'F2',TMPID,IERR)
      CALL ADFCRE(RID,'F3',PID,IERR)
      CALL ADFSLB(PID,'LABEL ON NODE F3',IERR)
      CALL ADFPDIM(PID,'R4',2,IDIMA,IERR)
      CALL ADFWALL(PID,A,IERR)
C
      CALL ADFCRE(PID,'F4',CID,IERR)
C
```

```
      CALL ADFCRE(PID,'F5',CID,IERR)
C
      CALL ADFDCLO(RID,IERR)
C
C *** 1.) OPEN 2ND DATABASE
C     2.) CREATE NODES
C     3.) PUT DATA IN N13
C
      CALL ADFDOPN('adf_file_one.adf','NEW',' ',RID,IERR)
C
C     THREE NODES UNDER ROOT
C
      CALL ADFCRE(RID,'N1',TMPID,IERR)
      CALL ADFCRE(RID,'N2',TMPID,IERR)
      CALL ADFCRE(RID,'N3',TMPID,IERR)
C
C     THREE NODES UNDER N1 (TWO REGULAR AND ONE LINK)
C
      CALL ADFGNID(RID,'N1',PID,IERR)
      CALL ADFCRE(PID,'N4',TMPID,IERR)
      CALL ADFLINK(PID,'L3','adf_file_two.adf','/F3',TMPID,IERR)
      CALL ADFCRE(PID,'N5',TMPID,IERR)
C
C     TWO NODES UNDER N4
C
      CALL ADFGNID(PID,'N4',CID,IERR)
      CALL ADFCRE(CID,'N6',TMPID,IERR)
      CALL ADFCRE(CID,'N7',TMPID,IERR)
C
C     ONE NODE UNDER N6
C
      CALL ADFGNID(RID,'/N1/N4/N6',PID,IERR)
      CALL ADFCRE(PID,'N8',TMPID,IERR)
C
C     THREE NODES UNDER N3
C
      CALL ADFGNID(RID,'N3',PID,IERR)
      CALL ADFCRE(PID,'N9',TMPID,IERR)
      CALL ADFCRE(PID,'N10',TMPID,IERR)
      CALL ADFCRE(PID,'N11',TMPID,IERR)
C
C     TWO NODES UNDER N9
C
      CALL ADFGNID(PID,'N9',CID,IERR)
      CALL ADFCRE(CID,'N12',TMPID,IERR)
      CALL ADFCRE(CID,'N13',TMPID,IERR)
C
C     PUT LABEL AND DATA IN N13
C
      CALL ADFSLB(TMPID,'LABEL ON NODE N13',IERR)
      CALL ADFPDIM(TMPID,'I4',1,IDIMC,IERR)
```

```
      CALL ADFWALL(TMPID,IC,IERR)
C
C     TWO NODES UNDER N10
C
      CALL ADFGNID(RID,'/N3/N10',PID,IERR)
      CALL ADFLINK(PID,'L1',' ','/N3/N9/N13',TMPID,IERR)
      CALL ADFCRE(PID,'N14',TMPID,IERR)
C
C     TWO NODES UNDER N11
C
      CALL ADFGNID(RID,'/N3/N11',PID,IERR)
      CALL ADFLINK(PID,'L2',' ','/N3/N9/N13',TMPID,IERR)
      CALL ADFCRE(PID,'N15',TMPID,IERR)
C
C *** READ AND PRINT DATA FROM NODES
C     1.) NODE F5 THROUGH LINK L3
C
      CALL ADFGNID(RID,'/N1/L3',PID,IERR)
      CALL ADFGLB(PID,TSTLBL,IERR)
      CALL ADFGDT(PID,DTYPE,IERR)
      CALL ADFGND(PID,NUMDIM,IERR)
      CALL ADFGDV(PID,IDIM,IERR)
      CALL ADFRALL(PID,B,IERR)
      PRINT *,' NODE F3 THROUGH LINK L3:'
      PRINT *,'   LABEL       = ',TSTLBL
      PRINT *,'   DATA TYPE   = ',DTYPE
      PRINT *,'   NUM OF DIMS = ',NUMDIM
      PRINT *,'   DIM VALS    = ',IDIM
      PRINT *,'   DATA:'
      WRITE(*,100)((B(J,I),I=1,3),J=1,4)
  100 FORMAT(5X,3F10.2)
C
C     2.) N13
C
      CALL ADFGNID(RID,'N3/N9/N13',PID,IERR)
      CALL ADFGLB(PID,TSTLBL,IERR)
      CALL ADFGDT(PID,DTYPE,IERR)
      CALL ADFGND(PID,NUMDIM,IERR)
      CALL ADFGDV(PID,IDIMD,IERR)
      CALL ADFRALL(PID,ID,IERR)
      PRINT *,' '
      PRINT *,' NODE N13:'
      PRINT *,'   LABEL       = ',TSTLBL
      PRINT *,'   DATA TYPE   = ',DTYPE
      PRINT *,'   NUM OF DIMS = ',NUMDIM
      PRINT *,'   DIM VALS    = ',IDIMD
      PRINT *,'   DATA:'
      WRITE(*,200)(ID(I),I=1,6)
  200 FORMAT(5X,6I6)
C
C     3.) N13 THROUGH L1
```

```
C
      CALL ADFGNID(RID,'N3/N10/L1',TMPID,IERR)
      CALL ADFGLB(TMPID,TSTLBL,IERR)
      CALL ADFRALL(TMPID,ID,IERR)
      PRINT *,' '
      PRINT *,' NODE N13 THROUGH LINK L1:'
      PRINT *,'   LABEL       = ',TSTLBL
      PRINT *,'   DATA:'
      WRITE(*,200)(ID(I),I=1,6)
C
C     4.) N13 THROUTH L2
C
      CALL ADFGNID(RID,'N3/N11/L2',CID,IERR)
      CALL ADFGLB(CID,TSTLBL,IERR)
      CALL ADFRALL(CID,ID,IERR)
      PRINT *,' '
      PRINT *,' NODE N13 THROUGH LINK L2:'
      PRINT *,'   LABEL       = ',TSTLBL
      PRINT *,'   DATA:'
      WRITE(*,200)(ID(I),I=1,6)
C
C     PRINT LIST OF CHILDREN UNDER ROOT NODE
C
      CALL PRTCLD(RID)
C
C     PRINT LIST OF CHILDREN UNDER N3
C
      CALL ADFGNID(RID,'N3',PID,IERR)
      CALL PRTCLD(PID)
C
C     REOPEN ADF_FILE_TWO AND GET NEW ROOT ID
C
      CALL ADFDOPN('adf_file_two.adf','OLD',' ',RID,IERR)
      PRINT *,' '
      PRINT *,' COMPARISON OF ROOT ID: '
      PRINT *,' ADF_FILE_TWO.ADF ORIGINAL ROOT ID = ',RIDF2
      PRINT *,' ADF_FILE_TWO.ADF NEW ROOT ID      = ',RID
C
      STOP
      END
C
C ************* SUBROUTINES ****************
C
      SUBROUTINE PRTCLD(PID)
C
C *** PRINT TABLE OF CHILDREN GIVEN A PARENT NODE-ID
C
      PARAMETER (MAXCLD=10)
      PARAMETER (MAXCHR=32)
      REAL*8 PID
      CHARACTER*(MAXCHR) NODNAM,NDNMS(MAXCLD)
```

```
      CALL ADFGNAM(PID,NODNAM,IERR)
      CALL ADFNCLD(PID,NUMC,IERR)
      WRITE(*,120)NODNAM,NUMC
  120 FORMAT(/,' PARENT NODE NAME = ',A,/,
     X       '     NUMBER OF CHILDREN = ',I2,/,
     X       '     CHILDREN NAMES:')
      NLEFT = NUMC
      ISTART = 1
C     --- TOP OF DO-WHILE LOOP
  130 CONTINUE
        CALL ADFCNAM(PID,ISTART,MAXCLD,LEN(NDNMS),
     X               NUMRET,NDNMS,IERR)
        WRITE(*,140)(NDNMS(K),K=1,NUMRET)
  140   FORMAT(8X,A)
        NLEFT = NLEFT - MAXCLD
        ISTART = ISTART + MAXCLD
      IF (NLEFT .GT. 0) GO TO 130
      RETURN
      END
```

The resulting output is:

```
NODE F3 THROUGH LINK L3:
  LABEL     = LABEL ON NODE F3
  DATA TYPE = R4
  NUM OF DIMS = 2
  DIM VALS  = 4 3
  DATA:
         1.10      1.20      1.30
         2.10      2.20      2.30
         3.10      3.20      3.30
         4.10      4.20      4.30

NODE N13:
  LABEL     = LABEL ON NODE N13
  DATA TYPE = I4
  NUM OF DIMS = 1
  DIM VALS  = 6
  DATA:
       1     2     3     4     5     6

NODE N13 THROUGH LINK L1:
  LABEL     = LABEL ON NODE N13
  DATA:
       1     2     3     4     5     6

NODE N13 THROUGH LINK L2:
  LABEL     = LABEL ON NODE N13
  DATA:
       1     2     3     4     5     6
```

```
PARENT NODE NAME = ADF MotherNode
    NUMBER OF CHILDREN =  3
    CHILDREN NAMES:
       N1
       N2
       N3

PARENT NODE NAME = N3
    NUMBER OF CHILDREN =  3
    CHILDREN NAMES:
       N9
       N10
       N11

COMPARISON OF ROOT ID:
  ADF_FILE_TWO.ADF ORIGINAL ROOT ID = 1.2653021189994324-320
  ADF_FILE_TWO.ADF NEW ROOT ID      = 4.7783097267391979-299
```

K C 语言程序样本

下面的程序建立的ADF文件见图1。

```
/*
   Sample ADF test program to build adf files illustrated
   in example database figure.
*/

#include <stdio.h>
#include <ctype.h>
#include <string.h>

#include "../include/ADF.h"

void print_child_list(double node_id);

main ()
{
  /* --- Node header character strings */
   char label[ ADF_LABEL_LENGTH+1];
   char data_type[ADF_DATA_TYPE_LENGTH+1];
   char file_name[ADF_FILENAME_LENGTH+1];
   char path[ADF_MAX_LINK_DATA_SIZE+1];

  /* --- Node id variables */
   double root_id,parent_id,child_id,tmp_id,root_id_file2;

  /* --- Data to be stored in database */
   float a[3][4] = {
                    1.1,2.1,3.1,4.1,
                    1.2,2.2,3.2,4.2,
                    1.3,2.3,3.3,4.3
                   };
   int a_dimensions[2] = {4,3};

   int c[6] = {1,2,3,4,5,6};
   int c_dimension = 6;

  /* --- miscellaneous variables */
   int error_flag, i, j;
   int error_state = 1;
   int num_dims, dim_d, d[6], dims_b[2];
   float b[3][4];

/* ------ begin source code ----- */

  /* --- set database error flag to abort on error */
   ADF_Set_Error_State(error_state,&error_flag);
```

```
/* -------- build file: adf_file_two.adf ---------- */
/* --- 1.) open database
       2.) create three nodes at first level
       3.) put label on node f3
       4.) put some data in node f3
       5.) create two nodes below f3
       6.) close database */

 ADF_Database_Open("adf_file_two.adf","new"," ",&root_id,&error_flag);
 root_id_file2 = root_id;
 ADF_Create(root_id,"f1",&tmp_id,&error_flag);
 ADF_Create(root_id,"f2",&tmp_id,&error_flag);
 ADF_Create(root_id,"f3",&parent_id,&error_flag);
 ADF_Set_Label(parent_id,"label on node f3",&error_flag);

 ADF_Put_Dimension_Information(parent_id,"R4",2,a_dimensions,&error_flag);
 ADF_Write_All_Data(parent_id,(char *)(a),&error_flag);

 ADF_Create(parent_id,"f4",&child_id,&error_flag);
 ADF_Create(parent_id,"f5",&child_id,&error_flag);
 ADF_Database_Close(root_id,&error_flag);

/* -------- build file: adf_file_one.adf ---------- */
/* open database and create three nodes at first level */
 ADF_Database_Open("adf_file_one.adf","new"," ",&root_id,&error_flag);
 ADF_Create(root_id,"n1",&tmp_id,&error_flag);
 ADF_Create(root_id,"n2",&tmp_id,&error_flag);
 ADF_Create(root_id,"n3",&tmp_id,&error_flag);

/* put three nodes under n1 (two regular and one link) */
 ADF_Get_Node_ID(root_id,"n1",&parent_id,&error_flag);
 ADF_Create(parent_id,"n4",&tmp_id,&error_flag);
 ADF_Link(parent_id,"l3","adf_file_two.adf","/f3",&tmp_id,&error_flag);
 ADF_Create(parent_id,"n5",&tmp_id,&error_flag);

/* put two nodes under n4 */
 ADF_Get_Node_ID(parent_id,"n4",&child_id,&error_flag);
 ADF_Create(child_id,"n6",&tmp_id,&error_flag);
 ADF_Create(child_id,"n7",&tmp_id,&error_flag);

/* put one nodes under n6 */
 ADF_Get_Node_ID(root_id,"/n1/n4/n6",&parent_id,&error_flag);
 ADF_Create(parent_id,"n8",&tmp_id,&error_flag);

/* put three nodes under n3 */
 ADF_Get_Node_ID(root_id,"n3",&parent_id,&error_flag);
 ADF_Create(parent_id,"n9",&tmp_id,&error_flag);
 ADF_Create(parent_id,"n10",&tmp_id,&error_flag);
 ADF_Create(parent_id,"n11",&tmp_id,&error_flag);

/* put two nodes under n9 */
```

```
 ADF_Get_Node_ID(parent_id,"n9",&child_id,&error_flag);
 ADF_Create(child_id,"n12",&tmp_id,&error_flag);
 ADF_Create(child_id,"n13",&tmp_id,&error_flag);

/* put label and data in n13 */
 ADF_Set_Label(tmp_id,"Label on Node n13",&error_flag);
 ADF_Put_Dimension_Information(tmp_id,"i4",1,&c_dimension,&error_flag);
 ADF_Write_All_Data(tmp_id,(char *)(c),&error_flag);

/* put two nodes under n10 (one normal, one link) */
 ADF_Get_Node_ID(root_id,"/n3/n10",&parent_id,&error_flag);
 ADF_Link(parent_id,"l1"," ","/n3/n9/n13",&tmp_id,&error_flag);
 ADF_Create(parent_id,"n14",&tmp_id,&error_flag);

/* put two nodes under n11 (one normal, one link) */
 ADF_Get_Node_ID(root_id,"/n3/n11",&parent_id,&error_flag);
 ADF_Link(parent_id,"l2"," ","/n3/n9/n13",&tmp_id,&error_flag);
 ADF_Create(parent_id,"n15",&tmp_id,&error_flag);

/* ================= finished building adf_file_one.adf ============= */

/* ============= access and print data ================ */

/* access data in node f3 (adf_file_two.adf) through link l3 */
 ADF_Get_Node_ID(root_id,"/n1/l3",&tmp_id,&error_flag);
 ADF_Get_Label(tmp_id,label,&error_flag);
 ADF_Get_Data_Type(tmp_id,data_type,&error_flag);
 ADF_Get_Number_of_Dimensions(tmp_id,&num_dims,&error_flag);
 ADF_Get_Dimension_Values(tmp_id,dims_b,&error_flag);
 ADF_Read_All_Data(tmp_id,(char *)(b),&error_flag);
 printf (" node f3 through link l3:\n");
 printf ("   label       = %s\n",label);
 printf ("   data_type   = %s\n",data_type);
 printf ("   num of dims = %5d\n",num_dims);
 printf ("   dim vals    = %5d %5d\n",dims_b[0],dims_b[1]);
 printf ("   data:\n");
 for (i=0; i<=3; i++)
   {
     for (j=0; j<=2; j++)
       {
         printf("     %10.2f",b[j][i]);
       };
     printf("\n");
   };

/* access data in node n13 */
 ADF_Get_Node_ID(root_id,"/n3/n9/n13",&tmp_id,&error_flag);
 ADF_Get_Label(tmp_id,label,&error_flag);
 ADF_Get_Data_Type(tmp_id,data_type,&error_flag);
 ADF_Get_Number_of_Dimensions(tmp_id,&num_dims,&error_flag);
 ADF_Get_Dimension_Values(tmp_id,&dim_d,&error_flag);
```

```
 ADF_Read_All_Data(tmp_id,(char *)(d),&error_flag);
 printf (" node n13:\n");
 printf ("   label       = %s\n",label);
 printf ("   data_type   = %s\n",data_type);
 printf ("   num of dims = %5d\n",num_dims);
 printf ("   dim val     = %5d\n",dim_d);
 printf ("   data:\n");
 for (i=0; i<=5; i++)
   {
     printf("     %-4d",d[i]);
   };
 printf("\n\n");

/* access data in node n13 through l1 */
 ADF_Get_Node_ID(root_id,"/n3/n10/l1",&tmp_id,&error_flag);
 ADF_Get_Label(tmp_id,label,&error_flag);
 ADF_Read_All_Data(tmp_id,(char *)(d),&error_flag);
 printf (" node n13 through l1:\n");
 printf ("   label       = %s\n",label);
 printf ("   data:\n");
 for (i=0; i<=5; i++)
   {
     printf("     %-4d",d[i]);
   };
 printf("\n\n");

/* access data in node n13 through l2 */
 ADF_Get_Node_ID(root_id,"/n3/n11/l2",&tmp_id,&error_flag);
 ADF_Get_Label(tmp_id,label,&error_flag);
 ADF_Read_All_Data(tmp_id,(char *)(d),&error_flag);
 printf (" node n13 through l2:\n");
 printf ("   label       = %s\n",label);
 printf ("   data:\n");
 for (i=0; i<=5; i++)
   {
     printf("     %-4d",d[i]);
   };
 printf("\n\n");

/* print list of children under root node */
 print_child_list(root_id);

/* print list of children under n3 */
 ADF_Get_Node_ID(root_id,"/n3",&tmp_id,&error_flag);
 print_child_list(tmp_id);

/* re-open adf_file_two and get new root id */
 ADF_Database_Open("adf_file_two.adf","old"," ",&root_id,&error_flag);
 printf (" Comparison of root id:\n");
 printf ("   adf_file_two.adf original root id = %x\n",root_id_file2);
 printf ("   adf_file_two.adf new      root id = %x\n",root_id);
```

```
}

void print_child_list(double node_id)
{

/*
   print table of children given a parent node-id
*/
   char node_name[ADF_NAME_LENGTH+1];
   int i, num_children, num_ret, error_return;

   ADF_Get_Name(node_id,node_name,&error_return);
   ADF_Number_of_Children(node_id,&num_children,&error_return);
   printf ("Parent Node Name = %s\n",node_name);
   printf ("  Number of Children = %2d\n",num_children);
   printf ("  Children Names:\n");
   for (i=1; i<=num_children; i++)
     {
       ADF_Children_Names(node_id,i,1,ADF_NAME_LENGTH+1,
           &num_ret,node_name,&error_return);
       printf ("     %s\n",node_name);                                  };
    printf ("\n");
}
```

The resulting output is:

```
node f3 through link l3:
  label      = label on node f3
  data_type  = R4
  num of dims =     2
  dim vals   =     4     3
  data:
          1.10           1.20           1.30
          2.10           2.20           2.30
          3.10           3.20           3.30
          4.10           4.20           4.30
node n13:
  label      = Label on Node n13
  data_type  = i4
  num of dims = 1
  dim val    = 6
  data:
    1       2       3       4       5       6

node n13 through l1:
  label      = Label on Node n13
  data:
    1       2       3       4       5       6

node n13 through l2:
  label      = Label on Node n13
```