SCIENTECH, INC.®

PIB File Specification

K. R. Jones

April 1997

Scientech Inc.
11140 Rockville Pike • Suite 500
Rockville, MD 20852
Phone 301/468-6425 • Fax 301/468-0883

Table of Contents

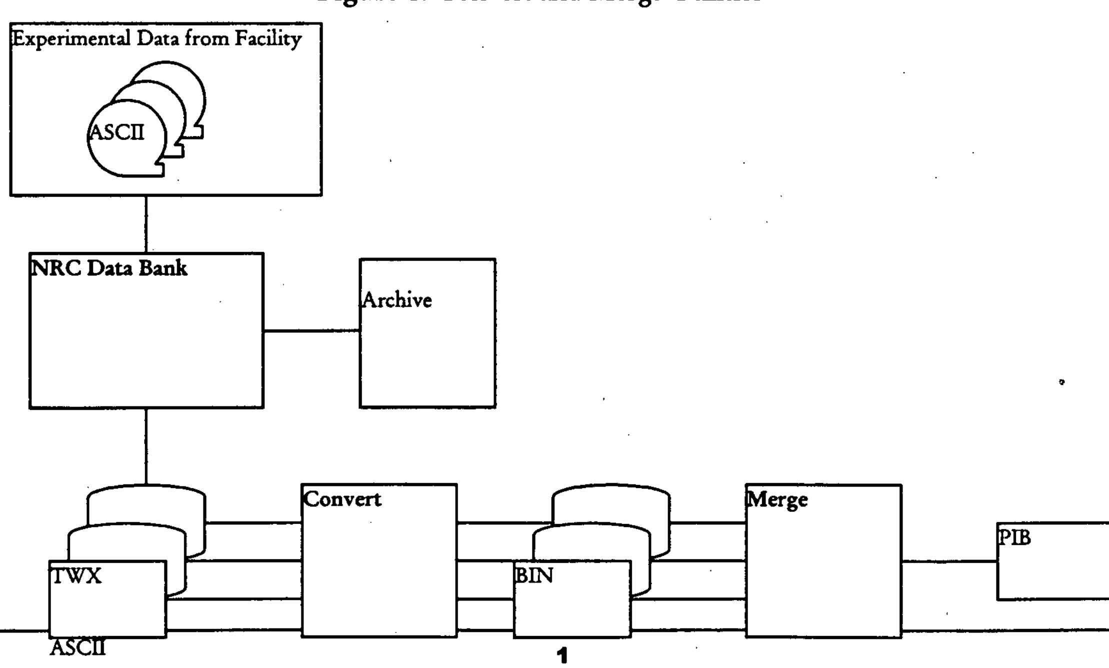
1. Overview

The Platform-Independent Binary or PIB file format was developed to provide a lightweight, machine-independent format for the plotting and analysis of data from the NRC Reactor Safety Data Dank. The NRC Data Bank was originally developed at the Idaho National Engineering Laboratory, to collect, store, and make available, data from many domestic and foreign light water reactor safety research programs. The NRC Data Bank is currently being modified and relocated to the NRC offices in Rockville, MD. The XMGR5 plotting and data analysis software is used to read this experimental data as well as calculated data generated from several reactor safety analysis programs including RELAP, MELCOR and FRAPCON.

The data files maintained by the NRC Data Bank consist of large time-dependent data files containing the results of experimental calculations. A single experiment can generate several hundred megabytes of data. ASCII files provide a portable data format for exchange of data between different machines, however, it is generally very inefficient and time consuming to store and extract data from large ASCII files. Native binary files can be used to store and retrieve data in a very efficient manner using direct access pointers, however, they cannot be read directly by other machine types. The PIB format uses the External Data Representation (XDR) protocol to provide a portable, machine-independent format that can be accessed with an efficiency approaching that of the native binary format.

Two utility programs, Convert and Merge were developed to provide a means of generating data files from the NRC Data Bank for XMGR5. Figure 1 illustrates the usage these programs as well as the role of the NRC Data Bank in the creation of PIB data files. Data is obtained from experimental facilities on magnetic tape. Each facility sends data in its own format, and a single experiment may encompass several tapes. These tapes are processed by the NRC Data bank and stored in a common binary format. Data stored in the NRC Data Bank can be dumped into an ASCII file using a standard format known as TWX. The Convert utility reads a single TWX file and converts it to a binary file that can be read by XMGR5. This BIN format was initially used to provide experimental data to XMGR5. Although it is possible for XMGR5 to read from several files simultaneously, it became clear that it would be desirable to merge all the data files from a single experiment into a single file that could then be read by XMGR5. Due to the wide range of Unix platforms used to analyze the data, there is also a benefit to switching to an efficient platform independent format. The Merge program can

Figure 1. Convert and Merge Utilities



read multiple BIN or PIB files and merge them into a single PIB file. In addition, the Merge program is used to verify data integrity, perform statistical analyses, apply digital filters, compress the channel data, and perform basic data management functions. Much of the example code shown in this report was derived from the Merge program.

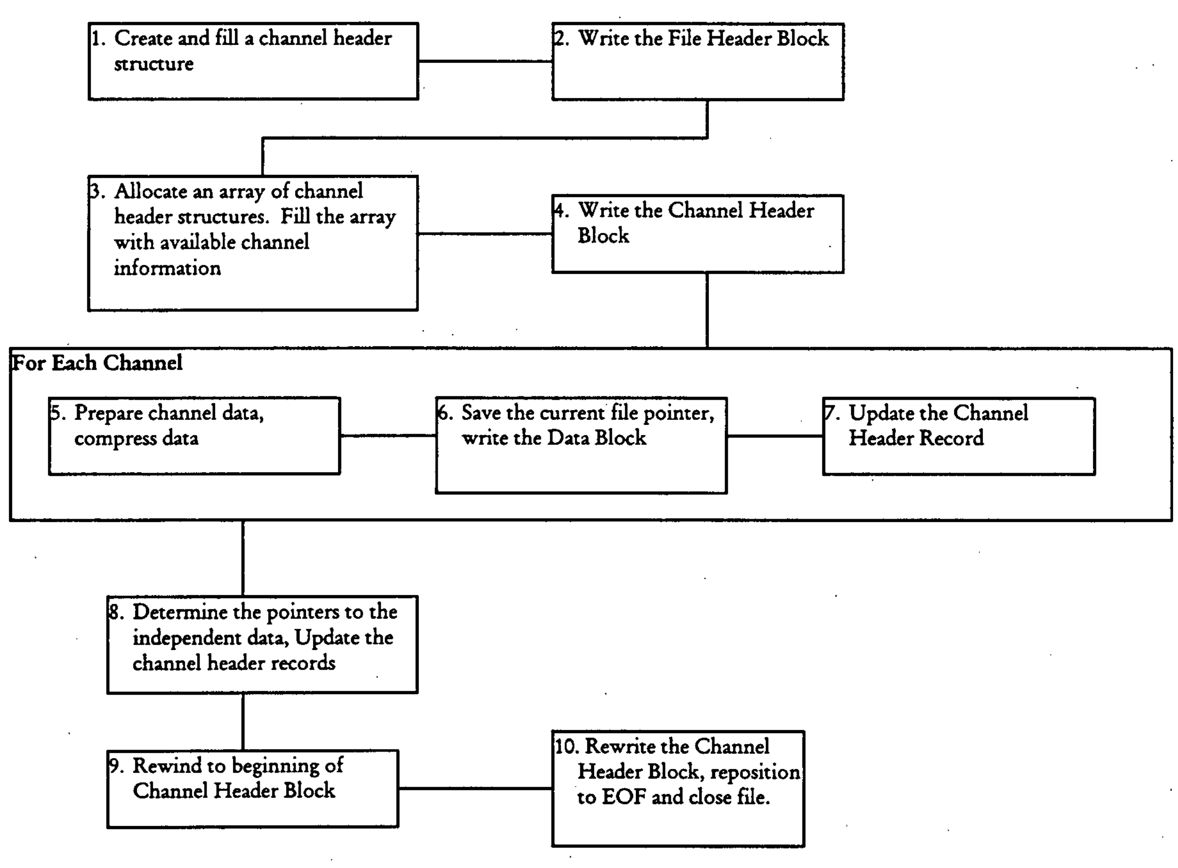
This document is meant to serve as a guide for creating PIB files. Several C-language code segments are provided to illustrate programming approaches. As such, a general knowledge of C-language programming is assumed.

1. File Structure

The PIB file structure is composed of three distinct sections, a file header block, a channel header block, and the channel data block. The file header block is used to identify the file type and it contains the information necessary to read the channel header block. The channel header block contains one channel header record for each channel in the file. The channel header records contain the information required to read the channel data block. A data channel can represent either a set of time values or a set of dependent data values. Each dependent data channel includes a reference to a time data channel that provides the independent data values. Many dependent data channels generally reference the same time channel.

In order to achieve platform independence, all data written to or retrieved from the PIB file is accessed by means of the External Data Representation (XDR) routines. In the UNIX operating system, the interface for the XDR primitives are defined in the include file <rpc/xdr.h>, which is automatically included by <rpc/rpc.h>. Refer to a programmers guide to remote procedure calls (RPC) for a description of the XDR protocol. One such reference is *Power Programming with RPC* by John Bloomer, O'Reilly & Associates, 1992.

Figure 2. PIB File Creation Process



The process of creating a PIB file is illustrated in Figure 2. As will be discussed further in subsequent sections, the PIB file is written in two passes. During the first pass, the file header block, the channel header block, and the channel data block are written to the file. However, the file pointer information required for the channel header block is not determined until the channel data block is written to file. For this reason, a second pass is required to update the channel header block.

The first two process steps involve creating the file header block structure and writing it to an XDR stream. These steps are discussed in Section 2.A, File Header Block. Steps 3, 4, 9 and 10, which involve writing the channel header block, are covered in Section 2.B. Steps 5, 6 and 7 involve writing the channel data block and updating the channel header records. These steps are discussed in Section 2.C.

A. File Header Block

The PIB file begins with a header block that is used to identify the file type and to provide the information necessary to read the subsequent channel header records. It may also include an optional list of filenames that were used to create the PIB file along with an integer value indicating their file type. Currently, integer values of 1000 and 2000 are reserved to identify binary and PIB file types respectively. This file list can prove useful in tracking down data sources, especially if several files are combined using the merge utility. Table 1 illustrates the layout of the file header block.

Table 1. File Header Block Layout

XDR Primitive	Description	Field Contents
xdr_string	File type string	"NRCDB V2.0, K. R. Jones"
xdr_int	Header size (currently not used)	0
xdr_int	Number of data channels	numOfChnls
xdr_int	Number of filenames (0 to 80)	numOfFiles
xdr_string	Name of 1st file. (optional)	fromfile[0]
xdr_string	Name of 2nd file. (optional)	fromfile[1]
...	...	...
xdr_string	Name of last file. (optional)	fromfile[numOfFiles]
xdr_int	Type of 1st file. (optional)	fromfileType[0]
xdr_int	Type of 2nd file. (optional)	fromfileType[1]
...	...	...
xdr_int	Type of last file. (optional)	fromfileType[numOfFiles]
xdr_string	Name of file being created	filename

The following xdr_fileHead routine can be used to read or write the file header block to an existing XDR stream. The header data is retrieved from or placed into a pibHeader structure as defined below:

Source Listing 1. File Header XDR Access Routine

```
struct pibHeader {
  char fileTyp[80];       /* string indicating file type */
  int size;               /* not used enter 0 */
  int numOfChnls;         /* the number of data channels contained in this file */
  int numOfFiles;         /* the number of data files in the file list */
  char fromfile[80][256];/* a list of up to 80 filenames */
  int fromfiletype[80];   /* corresponding file types */
  char tofile[256];       /* the name of the file being created */
};
```

```
bool_t xdr_fileHead(XDR *xdrs,  struct pibHeader *hdr)
{
   int i;
   bool_t rc;
   u_int slen;
   char *cptr;

   cptr = hdr->fileTyp;
   slen = 80;
   if(!(rc = xdr_string(xdrs,  &cptr, slen))) return rc;
   if(!(rc = xdr_int(xdrs, &hdr->size))) return rc;
   if(!(rc = xdr_int(xdrs, &hdr->numOfChnls))) return rc;
   if(!(rc = xdr_int(xdrs, &hdr->numOfFiles))) return rc;
   for(i=0; i<hdr->numOfFiles; i++) {
      cptr = &hdr->fromfile[i][0];
      slen = 256;
      if(!(rc = xdr_string(xdrs,  &cptr, slen))) return rc;
   }
   for(i=0; i<hdr->numOfFiles; i++) {
      if(!(rc = xdr_int(xdrs, &hdr->fromfiletype[i]))) return rc;
   }
   cptr = &hdr->tofile[0];
   slen = 256;
   if(!(rc = xdr_string(xdrs,  &cptr, slen))) return rc;
   return rc;
}
```

The following code segment demonstrates how to create the XDR stream and write the header using the xdr_fileHead routine. It assumes that the filename and numChan variables, corresponding to the name of the file to be created, and the number of data channels, respectively, have been previously defined.

Source Listing 2. Example of xdr_fileHead Usage

```
/* declarations */
XDR xdrs;
FILE fileHndl;
struct pibHeader hdr;
int rc;

/* copy data to the header structure */
strcpy(hdr->filetype, "NRCDB V2.0, K. R. Jones");
hdr->size = 0;
hdr->numOfChnls = numChan;
hdr->numOfFiles = 0;
strcpy(hdr->toFile, filename);

/* open the output file */
fileHndl=fopen(filename,"w");
if (fileHndl == NULL) {
   fprintf(stderr, "Unable to open file %s", filename);
   return;
};

/* create the XDR object */
xdrstdio_create(&xdrs, fileHndl, XDR_ENCODE);

/* write the header structure to the XDR stream */
rc = xdr_fileHead(&xdrs, &hdr);
if(!rc) {
   fprintf(stderr,"Error writting PIB file…");
   return;
}
```

B. Channel Header Block

The channel header block immediately follows the file header block. The channel header block contains one channel header record for each data channel. The layout of the channel header record is illustrated in Table 2 along with a description of each field. The channel name is stored in ASCII characters and must be padded to with NULL characters to a length of 24 bytes. This ensures a fixed size for the channel header record, allowing the channel header block to be read very efficiently. Each channel is assigned a unique zero based index to identify that is used to identify the channel.

Table 2. Channel Header Record Layout

XDR Primitive	Field Contents	Description
xdr_bytes	name[24]	The channel name. Must be NULL padded to 24 bytes.
xdr_int	Index	A zero based index used to identify the channel.
xdr_int	size	Number of data points for the channel.
xdr_int	totalSize	The length of the data in bytes. (8 * size)
xdr_int	timeIndex	The Index value for the channel containing the time values. Note: This value should be zero for time channels.
xdr_int	ptrToData	A pointer to the dependent data.
xdr_int	ptrToTime	A pointer to the independent (time channel) data.
xdr_int	eucode	An integer value indicating the engineering unit code for this channel. See Section 2, Engineering Unit Codes.
xdr_int	recNo	Reserved, enter 0.
xdr_int	orgIndex	The original sequence number used in the source file. (optional)
xdr_int	orgFile	A zero based index indicating the source file from the file list of the file header block. (optional)
xdr_int	status	Reserved, enter 0.
xdr_int	cmpMode	Compression mode. See Section 3, Compression.
xdr_int	cmpSize	Size of compressed data array.
xdr_int	spare1	Reserved, enter 0.
xdr_int	spare2	Reserved, enter 0.
xdr_int	spare3	Reserved, enter 0.

The most difficult fields to populate are the pointers to the data and time data, ptrToData and ptrToTime, respectively. These pointers are most easily determined using the xdr_getpos function prior to writing the channel data. Unfortunately, these values are not known at the time the channel header block is written. For this reason, the channel header block is written in two passes. During the first pass, write the file header block, the channel header block and then the data block. As the data block is written, use the xdr_getpos function to determine the file pointer prior to writing each data channel. After the data block has been written, rewind the file to the start of the channel header block and rewrite the channel header block.

The following xdr_chanHead routine can be used to read or write a single channel header record to an existing XDR stream. The channel data is retrieved from or placed into a pibChnlRec structure as defined below:

Source Listing 3. Channel Header XDR Access Routine

```
struct pibChnlRec {
  char name[24];
  int  Index;
  int  size;
  int  totalSize;
  int  timeIndex;
  int  ptrToData;
  int  ptrToTime;
  int  eucode;
  int  recNo;
  int  orgIndex;
  int  orgFile;
  int  status;
  int  cmpMode;
  int  cmpSize;
  int  spare1;
  int  spare2;
  int  spare3;
} ;

bool_t xdr_chanHead(XDR *xdrs, struct pibChnlRec *Chan)
{
   bool_t rc;
   u_int slen = 24;
   char *cptr;

   cptr = Chan->name;
   rc = (xdr_bytes(xdrs, &cptr, &slen, slen)
         && xdr_int(xdrs, &Chan->Index)
         && xdr_int(xdrs, &Chan->size)
         && xdr_int(xdrs, &Chan->totalSize)
         && xdr_int(xdrs, &Chan->timeIndex)
         && xdr_int(xdrs, &Chan->ptrToData)
         && xdr_int(xdrs, &Chan->ptrToTime)
         && xdr_int(xdrs, &Chan->eucode)
         && xdr_int(xdrs, &Chan->recNo)
         && xdr_int(xdrs, &Chan->orgIndex)
         && xdr_int(xdrs, &Chan->orgFile)
         && xdr_int(xdrs, &Chan->status)
         && xdr_int(xdrs, &Chan->cmpMode)
         && xdr_int(xdrs, &Chan->cmpSize)
         && xdr_int(xdrs, &Chan->spare1)
         && xdr_int(xdrs, &Chan->spare2)
         && xdr_int(xdrs, &Chan->spare3));

   return rc;
}
```

Typically, an array of pibChnlRec structures will be allocated and filled with the channel information, then the xdr_chanHead routine will be called once for each element of the array.

C. Channel Data Block

The channel data block follows immediately after the channel header block. It consists entirely of arrays of double precision floating point data output with the xdr routines. The channel data is written using the xdr_array primitive with a double precision element type. The following code segment illustrates writing the channel data block to an open XDR stream:

Source Listing 4. Channel Data Block Write

```
/* declarations */
u_int filePtr;
bool_t rc;
double *dptr;
u_int csize;
int i;
int totalNumChan = [number of channels];

for(i=0; i<totalNumChan; i++) {
   /* get the current file position
       note: save this to the channel header record array so you can write the
             file pointers in the second pass
   */
   filePtr = xdr_getpos(&xdrs);

   /* set up dptr and csize variables */
   dptr = [pointer to data];
   csize = [length of data array];

   /* compress the data */
   cmpMode =cmpres(dptr, &csize);

   /* output channel data */
   rc = xdr_array(&xdrs,          /* XDR stream */
                  (char **)&dptr, /* pointer to the data to be written */
                  (u_int *)&csize,/* number of elements */
                  csize,          /* maximum number of elements */
                  sizeof(double), /* element size */
                  xdr_double);    /* primative used to encode element */
   if(!rc) {
     fprintf(stderr,"Error writting output file. exiting");
     exit(-1);
   }

   /* now save the compression mode and data pointer to the
      channel header record array */
   chanHeader[i].ptrToData = filePtr;
   chanHeader[i].cmpMode = cmpMode;
}
```

After the data block has been written, and all of the channel data pointers determined, the pointers to the time data can be determined. This is accomplished by setting the time data pointer value, ptrTotime equal to the data pointer value, ptrToData for the corresponding time channel. For time channels, ptrToTime should be set equal to its own data pointer. This process is illustrated in the following code segment:

Source Listing 5. Setting Time Pointers

```
/* set pointers to time values */
for(i=0; i<totalNumChan; i++) {
  chanHeader[i].ptrToTime = -1;
  if(chanHeader[i].timeIndex == 0) {
    chanHeader[i].ptrToTime = chanHeader[i].ptrToData;
  }else{
    chanHeader[i].ptrToTime = chanHeader[chanHeader[i].timeIndex].ptrToData;
  }
  if(chanHeader[i].ptrToTime < 0) {
    fprintf(stderr, "No time data found for channel:%s\n",
            chanHeader[i].name);
    exit(-1);
  }
}
```

After the data block has been written, and the channel header record structures have been updated to include the compression and file pointer information, the file is rewound to the start of the channel header block, and the channel header block is overwritten.

1. Engineering Unit Codes

Each data channel must have an engineering unit code (eucode) defined in its channel header record corresponding to the physical units in which the data channel was calibrated. Table 3 contains a list of all available engineering unit codes. The description and units columns shown in Table 3 are used to provide axis labels in the plotting software. These codes were originally developed for the NRC Data Bank to provide a common set of units to handle data from a wide range of domestic and foreign research programs.

In the event that none of the available unit codes are suitable for a given channel and an engineering unit code must be added, contact the NRC Project Manager for the NRC Data Bank to reserve a unit code. This will prevent conflicting use of a code value and ensure that the appropriate modifications are made to the plotting and analysis software.

Table 3. Engineering Unit Codes

Eng. Unit Code	Description	Units
1	Core Heater Temperature	F
2	Fluid Temperature	F
3	Pressure	psig
4	Strain	
5	Volumetric Flow	gpm
6	Fluid Velocity	ft/s
7	Force	lb
8	Length	in
9	Voltage	
10	Material Temperature	F
11	Current	Amp
12	Specific Volume	ft^3/lbm
13	Decibels	dB
14	Pressure	psi
15	Pressure	psia
16	Differential Pressure	psid
17	Density	lbm/ft^3
18	Power	kW
19	Heat Flux	Btu/s*ft^2
20	H. T. Coeff.	Btu/s*ft^2*F
21	Surface Temperature	F
22	Saturation Temperature	F
23	Enthalpy	Btu/lbm
24	Mass Flux	lbm/s*ft^2
25	Mass Flow	lbm/s
26	Integrated Mass Flow	lbm
27	Momentum Flux	lbm/ft*s^2
28	Fluid Velocity	ft/s
29	Pump Speed	rpm
30	Elevation	ft
31	Quality	
32	Normalized Power	
33	Mass Flux	10e6 lbm/hr*ft^2
34	Temperature	F
35	Time After Rupture	s
36	Time	s
37	Total Energy	Btu
38	Reactivity	$
39	Stored Energy	Btu
40	Energy	Btu
41	Mass Balance	lbm
42	Power	MW

Eng. Unit Code	Description	Units
43	Total Heat Removed	Btu/s
44	Period	s
45	Heat Transfer Rate	Btu/s
46	Mass	lbm
47	Saturation Pressure	psia
48	Normalized Pump Torque	N*m
49	Volumetric Flow	ft^3/s
50	Choking Index	
51	Heat Transfer Mode	
52	Time After Reflood	s
53	Thermal Conductivity	Btu/s*ft*F
54	Internal Rod Temperature	F
55	Liquid Level	in
56	Percent	
57	Frequency	Hz
58	Total Volume	ft^3
59	Acceleration	ft/s^2
60	Core Heater Temperature	K
61	Fluid Temperature	K
62	Pressure	kPa
63	Strain	mm/m
64	Volumetric Flow	l/s
65	Fluid Velocity	m/s
66	Force	N
67	Length	cm
68	Material Temperature	K
69	Specific Volume	m^3/kg
70	Differential Pressure	kPa
71	Density	kg/m^3
72	Heat Flux	W/m^2
73	H. T. Coeff.	W/m^2*K
74	Surface Temperature	K
75	Saturation Temperature	K
76	Enthalpy	J/kg
78	Mass Flux	kg/s*m^2
79	Mass Flow	kg/s
80	Integrated Mass Flow	kg
81	Momentum Flux	kg/m*s^2
82	Fluid Velocity	cm/s
83	Elevation	m
84	Temperature	K
85	Time After Rupture	s
86	Time	s
87	Pressure	MPa
88	Time After Reflood	s
89	Angular Velocity	rad/s
90	Pump Torque	N*m
91	Liquid Level	cm
92	Thermal Conductivity	kW/m*K
93	Internal Rod Temperature	K
94	Volumetric Flow	ml/s
95	Void Fraction	
96	Temperature Difference	K
97	Photo Tube Temperature	K
98	Average Velocity	ft/s
99	Liquid Phase Velocity	ft/s
100	Vapor Phase Velocity	ft/s
101	Horsepower	kW
102	Mass Flow / Vol	lbm/ft^3*s
103	Slip Ratio	
104	Flow Quality	
105	Thermodynamic Quality	
106	Steam Quality	
107	Neutron Detectors	

Eng. Unit Code	Description	Units
108	Valve Position	
109	Valve Position	
110	Guide Tube Temperature	F
111	Fuel Rod Temperature	F
112	Reactor Power	MW
113	Fuel Rod Peak Power	kW/m
114	Fuel Rod Ave Power	kW/m
115	S-P Neutron Detector Curr	na
116	Neutron Flux	n/cm^2*s
117	Fuel Off-Center Temperature	K
118	Fuel Centerline Temperature	K
119	Outlet Temperature	K
120	Inlet Temperature	K
121	Cladding Elongation	mm
122	Cladding Elongation	
123	Rod Internal Pressure	MPa
124	Peak Flux	n/cm^2*s
125	Cladding Surface Temperature	K
126	Momentum Flux	10e3 lbm/ft*s^2
127	Total Density	kg/m^3
128	Liquid Density	kg/m^3
129	Vapor Density	kg/m^3
130	Specific Int Energy	J/kg
131	Specific Liq Int Energy	J/kg
132	Specific Vap Int Energy	J/kg
133	Liquid Void Fraction	
134	Vapor Void Fraction	
135	Volume Liquid Velocity	m/s
136	Volume Vapor Velocity	m/s
137	Volume Pressure	Pa
138	Volume Static Quality	
139	Volume Equilibrium Quality	
140	Volume Heat Source	W
141	Volume Liquid Temperature	K
142	Volume Vapor Temperature	K
143	Volume Equil Temperature	K
144	Volume Sonic Velocity	m/s
145	Junction Liq Velocity	m/s
146	Junction Vap Velocity	m/s
147	Interface Velocity	m/s
148	Junction Liq Density	kg/m^3
149	Junction Vap Density	kg/m^3
150	Junction L/I Energy	J/kg
151	Junction V/I Energy	J/kg
152	Power Input	W
153	Heat Transfer Rate	W
154	Critical Heat Flux	W/m^2
155	Heat Transfer Coef	W/m^2*K
156	Mesh Point Temperature	K
157	Mass Flow Rate	kg/s
158	Viscosity	lbm/ft*hr
159	Viscosity	cp
160	Liquid Viscosity	lbm/ft*hr
161	Liquid Viscosity	cp
162	Vapor Viscosity	lbm/ft*hr
163	Vapor Viscosity	cp
164	Surface Tension	lbf/ft
165	Surface Tension	N/m
166	Specific Heat	btu/lbm*F
167	Specific Heat	J/kg*K
168	Liquid Specific Heat	btu/lbm*F
169	Liquid Specific Heat	J/kg*K
170	Vapor Specific Heat	btu/lbm*F
171	Vapor Specific Heat	J/kg*K

Eng. Unit Code	Description	Units
172	Heat of Vaporization	Btu/lbm
173	Heat of Vaporization	kJ/kg
174	Thermal Diffusivity	ft^2/s
175	Thermal Diffusivity	m^2/s
176	Time	
177	Time After Rupture	
178	Time To CHF	
179	Crit. Heat Flux	btu/hr*ft^2
180	Crit. Heat Flux	kW/m^2
181	Power	btu/hr
182	Vapor Velocity	ft/s
183	Vapor Velocity	m/s
184	Flooding Rate	ft/s
185	Flooding Rate	m/s
186	LEIDENFROST Temperature	F
187	LEIDENFROST Temperature	K
188	T[wall] - T[sat]	F
189	T[wall] - T[sat]	K
190	Distance	ft
191	Distance	m
192	Area	ft^2
193	Area	m^2
194	Area	in^2
195	Area	cm^2
196	Diameter	ft
197	Diameter	m
198	Diameter	in
199	Diameter	cm
200	Radius	ft
201	Radius	m
202	Radius	in
203	Radius	cm
204	Volume	ft^3
205	Volume	m^3
206	Discharge Coefficient	
207	Flow Regime	
208	Friction Factor	
209	REYNOLDS NUMBER	
210	WEBER NUMBER	
211	LEWIS NUMBER	
212	FROUDE NUMBER	
213	KNUDSEN NUMBER	
214	STABILITY NUMBER	
215	NUSSELT NUMBER	
216	PRANDTL NUMBER	
217	MARTINELLI NUMBER	
218	BOILING NUMBER	
219	MACH NUMBER	
220	GRASHOF NUMBER	
221	RALEIGH NUMBER	
222	STANTON NUMBER	
223	ECKERT NUMBER	
224	EULER NUMBER	
225	STROUHAL NUMBER	
226	Liquid Density	lbm/ft^3
227	Vapor Density	lbm/ft^3
228	Power	kW/m
229	Mass	kg
230	Current	amp
231	Counts	log[c/s]
232	Density	mg/m^3
233	Momentum Flux	mg/m*s^2
234	Voltage	V
235	Velocity	m/s

Eng. Unit Code	Description	Units
236	Specific Entropy	btu/lbm*R
237	Specific Entropy	kJ/kg*K
238	Delta-Theta	rad
239	Pump Head	m^2/s^2
240	Pump Momentum Source	m/s^2
241	Volumetric Flow Rate	m^3/s
242	Temperature	C
243	Enthalpy	GJ
244	Enthalpy Flow	MW
245	Mass	mg
246	Mass	kg
247	Depressurization Rate	kPa/s
248	Saturation Temperature	C
249	Liquid Level	m
250	CP SECONDS SMALL JOB CLASS	s
251	CP SECONDS MEDIUM JOB CLASS	s
252	CP SECONDS LARGE JOB CLASS	s
253	CP SECONDS ELEPHANT JOB CLASS	s
254	I/O SECONDS SMALL JOB CLASS	s
255	I/O SECONDS MEDIUM JOB CLASS	s
256	I/O SECONDS LARGE JOB CLASS	s
257	I/O SECONDS ELEPHANT JOB CLASS	s
258	TOTAL CP TIME	s
259	TOTAL I/O TIME	s
260	JULIAN DAY	
261	INTERCON CP TIME	s
262	INTERCOM I/O TIME	s
263	SYSTEM SECONDS	ss
264	Accumualted CP Seconds	s
265	Accumualted I/O Seconds	s
266	Drag Disk	mv
267	Valve Position	mv
268	Level	mv
269	RHOF	lbm/ft^3
270	RHOG	lbm/ft^3
271	RHOL	lbm/ft^3
272	Current	ka
273	Differential Pressure	MPa
274	Cladding Temperature	K
275	Metal Temperature	K
276	Local Heat Generation	kW/m
277	Fluid Density	mg/m^3
278	Coolant Temperature	K
279	Guide Tube Temperature	K
280	Displacement	mm
281	Pump Power	kW
282	Power	%
283	Fluid Subcooling	K
284	Differential Pressure	Pa
285	Rod Position	m
286	Saturation Pressure	Pa
287	Saturation Pressure	kPa
288	Saturation Pressure	MPa
289	Average Density	mg/m^3
290	Average Pressure	MPa
291	Average Pressure	kPa
292	Average Temperature	K
293	Average Velocity	m/s
294	Ave Momentum Flux	mg/m*s^2
295	Power	np
296	Pump Torque	lbf*ft
297	Pump Torque	%
298	Mass Flow	lbm/hr
299	Current	mA

Eng. Unit Code	Description	Units
300	Voltage	mV
301	Fuel Rod Average Power	kW/ft
302	Distance	mm
303	Volume	mm^3
304	Volume	in^3
305	Energy	J/kg
306	Mass Flux	lb/hr*ft^2
307	Distance	mil
308	Gas Flow Rate	gm*moles/s
309	Total Energy	J
310	Strain	microm/m
311	Displacement	in
312	Current	log[A]
313	Potential	V
314	Displacement	m
315	Reactor Power	GW
316	Displacement	cm
317	Time (s from year 1900)	s
318	Displacement	in
319	ROUHANI Liquid Velocity	m/s
320	ROUHANI Vapor Velocity	m/s
321	AYA Liquid Velocity	m/s
322	AYA Vapor Velocity	m/s
323	Volumetric Liquid Velocity	m/s
324	Volumetric Vapor Velocity	m/s
325	Local Heat Generation	kW/ft
326	Temperature Difference	K
327	Temperature Difference	C
328	Mass Flow Rate	lbm/s
329	Coolant Temperature	F
330	Cladding Temperature	F
331	Fluid Subcooling	F
332	Differential Pressure	in
333	Volumetric Flow Rate	l/s
334	Power	kW/m
335	Energy	MW*hr
336	Neutron Detectors	nano amps
337	Fission Product Detectors	counts
338	Heat Flux	btu/s*ft^2
339	H. T. Coeff.	btu/s*ft^2*F
340	Metal Temperature	F
341	Average Density	mg/m^3
342	Fluid Density	mg/m^3
343	Mass Velocity	lbm/hr*ft^2
344	Inlet Subcooling	btu/lbm
345	Length	ft
346	Valve Position	V
347	Pressure	Pa
348	Differential Pressure	mmwg
349	Volumetric Flow	m^3/hr
350	Boron Concentration	ppm
351	Reactor Power	W
352	Rotations	rad
353	G's/Radian	
354	Radians	
355	G's	
356	Moments	lbf*in
357	Moments	N*m
358	Absolute Pressure	kg/m*s^2
359	Differential Pressure	kg/m*s^2
360	Rotation Speed	m/s
361	Event	
362	Pressure	bar
363	Differential Pressure	bar

Eng. Unit Code	Description	Units
364	Time	min
365	Mass Flow	lbm/hr
366	Differential Pressure	mb
367	EDQ	
368	IQF	
369	Fuel Plenum Temperature	K
370	Fuel Temperature	K
371	Power	GW
372	Neutron Flux	10X13 n/cm^2*s
373	Power	kW/ft
374	Tank Level	l
375	Neutron Detector	W/cm
376	Fuel Axial Strain	%
377	Cladding Axial Strain	%
378	Rod Internal Pressure	psia
379	Fuel Centerline Temperature	C
380	Cladding Circ Strain	%
381	Gap Conductance	Btu/hr*ft^2*F
382	Cladding Surface Temp	C
383	Mass Flow Rate	mlbm/hr
384	Vol Nuc Heat Power	W/m^3
385	Differential Pressure	m-h2o
386	Pressure	kg/cm^2
387	Flow Rate	kg/hr
388	Heat Flux	W/m^2
389	Concentration	mg/kg
390	Concentration	ppm
391	Conductivity	mu*mno/cm
392	Oxidation-Reduction-Pot	mV
393	Alkalinity (as CaCO3)	mg/kg
394	Calculated Diff Pressure	in h2o
395	Volumetric Flow Rate	gpm
396	Outlet Temperature	K
397	Mass Flow	kg/s
398	Pressure	MPa
399	Pressure	MPa
400	Frequency	Hz
401	Percent	%
402	Distance	um
403	Heat	Nm
404	Enthalpy Flow	kW
405	Distance	m
406	Pressure	KPa
407	Pressure	KPa
408	Mass Flow	kg/s
409	Density	kg/m^3
410	Outlet Temperature	channel
411	Mass Flux	kg/s*m^2
412	Distance	mm
413	Mass	kg
414	Outlet Temperature	uV
415	Pressure	Pa
416	Voltage	kg/cm^2
417	Intercom I/O Time	sec
420	Void Fraction - Cond.Probe	%
421	Average Density	kg/m^3
422	Beam Density	kg/m^3
423	Densitometer Output	V
424	Mass Flow Rate into Tank	kg/s
425	Average Mass Flow Rate	kg/s
426	Diff. Pressure-Liquid Level	KPa
427	Heat Transfer Rate	MW
428	Fluid Mass in Component	kg
429	Drag Disk Output	V

Eng. Unit Code	Description	Units
430	Pump Speed	Hz
431	Vibration Amplitude (rms)	micro m
432	Pitot Tube Output	V
433	Valve Pos. Control Signal	%
434	Pitot Tube Location	mm
435	Heated Thermocouple Output	K
436	HTC Probe Output	uV
437	Wall Temperature Output	V
438	Pitot Tube DP	kg/m*s^2
439	Pitot Tube Location	mm
440	Int. Catch Tank Disc. Flow	kg
441	Volumetric Flow Rate	scfm
442	Limit Switch Position	
443	Unknown	
444	Volumetric Flow (ACFM)	ft^3/min
445	Unknown	
446	Unknown	
447	Unknown	
448	Unknown	
449	Unknown	
450	Unknown	

1. Compression

The PIB data is compressed on an individual channel basis, as opposed to full file compression. This approach provides an efficient method of data retrieval, permitting direct access to the channel data and eliminating the need to uncompress the entire file prior to extracting the data for an individual channel. The compression method used is defined by the compression mode or cmpMode entry of the channel header record. Currently, three compression modes are available as indicated in Table 4.

Table 4. Channel Compression Modes

Compression Mode	Description
0	No compression used.
1	Flat data channel. Single value written to file.
2	Double precision run length encoded compression.

Typically, compression is turned off for a channel (cmpMode = 0) if the achievable compression falls below 5%. This arbitrary threshold balances the potential savings in storage requirements against the overhead associated with uncompressing the data.

If the data channel does not vary over the entire range of time, a compression mode of cmpMode=1 is used. A single value is then stored in the channel data block to represent the entire range of data.

The double precision run length encoding compression method is typically used if the achievable compression exceeds 5%. Although it is a very simple algorithm, significant compression is achievable with very little impact on performance. In this method, each set of identical, consecutive values is replaced by two values; the first value being the number of values replaced and the second being the actual value. Regions that are not compressed are preceded by a negative value that indicates the length of the uncompressed region.

Table 5 illustrates application of the algorithm to a set of raw values. The initial two values 518.3 and 518.4 are replaced by the set of values -2.0, 518.4 and 518.4. The -2.0 value indicates that the next two values are not compressed. This is followed by the value 518.5 repeated twelve times which is replaced by the pair 12.0, 518.5. Similarly, the next four values have no repeats and are preceded by a value of -4.0, while the last eight values repeat and are replaced by the pair 8.0, 518.9.

Table 5. Example of Run Length Compression

Raw Data	Compressed Data
518.3	-2.0
518.4	518.3
518.5	518.4
518.5	12.0
518.5	518.5
518.5	-4.0
518.5	518.6
518.5	518.9
518.5	518.6
518.5	518.8
518.5	8.0
518.5	518.9
518.5	
518.5	
518.6	
518.9	
518.6	
518.8	
518.9	
518.9	
518.9	
518.9	
518.9	
518.9	
518.9	
518.9	

The following two routines can be used to compress and uncompress the data, respectively.

Source Listing 6. Compression Routines

```
/* define a temporary storage array used for compression */
#define DBUFSIZE 60000
static double flbuf[DBUFSIZE];

/*  Function: cmpres
 *    Purpose: compress an array of double precision numbers
 * Arguments:
 *     data on entry contains raw channel data
 *          on exit contains compressed channel data
 *     size on entry contains length of raw channel data
 *          on exit contains length of compressed channel data
 *   Returns:
 *     compression mode
 *      0 = no compression
 *      1 = flat channel
 *      2 = run length compression
 */
int cmpres(double **data, int *size)
{
  int i,j;
  int cm;
  int reps,difs,dpos;

  if(*size > DBUFSIZE) {
    fprintf(stderr,"Channel length, %d, exceeds DBUFSIZE",*size);
```

```
    exit (-1);
  }

  cm = 0;
  j = 0;

  dpos = -1;
  reps = 0;
  difs = 0;
  for(i=1; i<*size; i++) {
    if(data[0][i] == data[0][i-1]) {
      if(difs) {
        flbuf[dpos] = (double)(-1*difs);
        difs = 0;
        dpos = -1;
      }
      reps++;
    } else {
      if(reps) {
        reps++;
        flbuf[j++] = (double)reps;
        flbuf[j++] = data[0][i-1];
        reps = 0;
      } else {
        if(dpos == -1) {
          dpos = j;
          j++;
        }
        flbuf[j++] = data[0][i-1];
        difs++;
      }
    }
  }
  if(reps) {
    reps++;
    flbuf[j++] = (double)reps;
    flbuf[j++] = data[0][i-1];
  } else if(difs) {
    flbuf[j++] = data[0][i-1];
    difs++;
    flbuf[dpos] = (double)(-1*difs);
  } else {
    flbuf[j++] = 1.0;
    flbuf[j++] = data[0][i-1];
  }
  if(j >= 0.95* (*size)) {
    cm = 0;
  } else if (reps == *size) {
    *size = 1;
    cm = 1;
  } else  {
    cm = 2;
    *size = j;
    for(i=0; i<*size; i++) {
      data[0][i] = flbuf[i];
    }
  }
  return cm;
}
```

```
/*  Function: uncmpres
 *   Purpose: uncompress an array of double precision numbers
 * Arguments:
 *     data on entry contains compressed channel data
 *          on exit contains raw channel data
 *   dblbuf pointer to temporary array space
 *    csize on entry contains length of compressed channel data
 *    fsize on entry contains length of raw channel data
 *       cm on entry contains compression mode
 *             0 = no compression
 *             1 = flat channel
 *             2 = run length compression
 *   Returns:
 *      0
 */
int uncmpres(double *data, double *dblbuf, int* csize, int fsize, int cm)
{
  int  i,j,k;
  int difs, reps;

  switch (cm) {
  case 0:
    break;     /* no compression */
  case 1:
    for(i = 0; i<fsize; i++) {
     data[i] = data[0];
    }
    break;     /* flat */
  case 2:
    k = i = 0;
    while(i<*csize) {
      if(data[i] < 0.0) {
        difs = (int)( -1*data[i] + 0.1 );
         i++;
         for(j = 0; j<difs; j++) {
           dblbuf[k++] = data[i++];
         }
      } else if(data[i] > 0.0) {
        reps = (int)( data[i] + 0.1 );
        for(j = 0; j<reps; j++) {
           dblbuf[k++] = data[i+1];
        }
        i+=2;
      } else {
        return 1; /* Error uncompressing data cm=2 */
      }
    }
    if(k != fsize) {
      return 2;  /* Error uncompressing data cm=2 */
    }
    for(i = 0; i<fsize; i++) {
     data[i] = dblbuf[i];
    }
    break;  /* run length compression */
  }
  return 0;
}
```