```
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++

Copyright (c) 2001 BeComm Corporation

Filename:

    fanout.c

Group Name:

    todo

Group Overview:

    Input variables:

    Fanout/Count   : int specifies the number of output paths.
                     if not set uses length of Fanout/KeyList

    Fanout/KeyList : array specifies a list of values to be used as keys
                     if not set no keys will be set.

    Fanout/KeyName : string specifies the path context name to set the key
at.
                     If not set, defaults to "Fanout/Key"

    Output variables:

    Fanout/Index   : int 0-based index of path

    Fanout/Key
    or value of
    Fanout/KeyName : object from Fanout/KeyList array

    Caller must set at least one of Fanout/Count and Fanout/KeyList.

Owner:

    Guy Carpenter (guyc) 28-Sep-2001

--------------------------------------------------------------------------
---*/

#define SOS_DEBUG_ZONE "/beads/fanout"
```

```
43
44 #include <sosstrings.h>
45 #include <sosmultimedia.h>
46
47 SOS_SOURCE_VERSION (
48     "$Id: fanout.c,v 1.4 2001/11/10 00:51:14 guyc Exp $"
49 );
50
51 /*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
…  +++++
52 Named Constants
53 ----------------------------------------------------------------------
…  ---*/
54
55 /*
56  * Name of bead
57  */
58 static const char BEAD_NAME[]        = "fanout";
59 static const char INPUT_COUNT[]      = "Fanout/Count";
60 static const char INPUT_KEYARRAY[]   = "Fanout/KeyList";
61 static const char INPUT_KEYNAME[]    = "Fanout/KeyName";
62 static const char OUTPUT_INDEX[]     = "Fanout/Index";
63 static const char DEFAULT_KEYNAME[] = "Fanout/Key";
64
65 /*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
…  +++++
66 Structs
67 ----------------------------------------------------------------------
…  ---*/
68
69 typedef struct {
70     SOS_UINT32            PathCount;
71     SOS_PATH **           PathList;
72 } FANOUT_CONTEXT;
73
74 /*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
…  +++++
75 Context Stuff
76 ----------------------------------------------------------------------
…  ---*/
77
78 static
79 void
80 FanOut_ContextDestroy(
    FANOUT_CONTEXT *        Context
```

```
82  )
83  {
84      SOS_DEBUGOUT_FUNC_TRACE("FanOut_ContextDestroy\n");
85
86      if (Context) {
87          if (Context->PathList) {
88              SOS_UINT32 i;
89              for (i=0;i<Context->PathCount;i++) {
90                  SOS_Path_Destroy(Context->PathList[i]);
91              }
92              SOS_Mem_Free(Context->PathList);
93          }
94          SOS_Mem_Free(Context);
95      }
96  }
97
98  static
99  FANOUT_CONTEXT *
100 FanOut_ContextCreate(
101     void
102 )
103 {
104     FANOUT_CONTEXT *context;
105
106     SOS_DEBUGOUT_FUNC_TRACE("FanOut_ContextCreate\n");
107
108     context = SOS_Mem_Alloc(sizeof(*context));
109     if (context) {
110         SOS_memset(context, 0, sizeof(*context));
111     }
112
113     return context;
114 }
115
116 static
117 SOS_STATUS
118 FanOut_ContextInit(
119     FANOUT_CONTEXT *     Context,
120     SOS_PATH *           ParentPath,
121     SOS_UINT32           PathCount,
122     SOS_IARRAY *         KeyArray,
123     const char *         KeyName
124 )
125 {
```

```
127     SOS_UINT32 i;
128
129     Context->PathCount = PathCount;
130     Context->PathList = SOS_Mem_Alloc(sizeof(SOS_PATH*)*PathCount);
131
132     if (Context->PathList) {
133         SOS_memset(Context->PathList, 0, sizeof(SOS_PATH*)*PathCount);
134         for (i=0;i<PathCount && SOS_SUCCEEDED(status);i++) {
135             Context->PathList[i]=SOS_Path_Create(ParentPath);
136             if (Context->PathList[i]) {
137
138                 SOS_REGOBJECT *indexObject = SOS_UInt32_Create(i);
139                 SOS_REGOBJECT *key;
140
141                 if (indexObject) {
142                     SOS_Path_AttributeSet(
143                         Context->PathList[i],
144                         OUTPUT_INDEX,
145                         indexObject
146                     );
147                     SOS_RegObject_Release(indexObject);
148                 }
149
150                 if (KeyArray &&
… SOS_SUCCEEDED(KeyArray->Get(KeyArray,i,&key))) {
151                     SOS_Path_AttributeSet(
152                         Context->PathList[i],
153                         KeyName,
154                         key
155                     );
156                     SOS_RegObject_Release(key);
157                 }
158             } else {
159                 status = SOS_Error;
160             }
161         }
162     } else {
163         /* no pathlist */
164         status = SOS_Error;
165     }
166
167     return status;
168 }
169
```

```
171 SOS_STATUS
172 FanOut_MessageSend(
173     FANOUT_CONTEXT *      Context,
174     SOS_MESSAGE *         Message
175 )
176 {
177     SOS_STATUS status = SOS_Success;
178 
179     if (Context->PathList) {
180         SOS_UINT32 i;
181         for (i=0;i<Context->PathCount && SOS_SUCCEEDED(status);i++) {
182             SOS_MESSAGE *copy;
183             SOS_Message_HeadMessageCopyFrom(Message,SOS_UINT32_MAX,
…&copy);
184             status = SOS_Path_MessageSend(Context->PathList[i], copy);
185         }
186     } else {
187         status = SOS_Error;
188     }
189     return status;
190 }
191 
192 /*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
…+++++
193 
194 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
…+++*/
195 
196 static
197 SOS_STATUS
198 FanOut_KeyCreate(
199     SOS_PATH     *Path,
200     SOS_MESSAGE  *Message
201 )
202 {
203     SOS_STATUS status = SOS_Success;
204     static SOS_UINT32 s_UniqueId = 0;
205     SOS_REGOBJECT*  uniqueSessionKey;
206 
207     SOS_DEBUGOUT_FUNC_TRACE("FanOut_KeyCreate\n");
208 
209     uniqueSessionKey = SOS_UInt32_Create(s_UniqueId++);
210     SOS_Path_SessionKeySet(Path, uniqueSessionKey);
211     SOS_RegObject_Release(uniqueSessionKey);
```

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.