```
int
rt_annotation_import4(struct rt_db_internal *ip, const struct bu_external *ep, const fastf_t
*mat, const struct db_i *dbip)
{
    struct rt_annotation_internal *annotation_ip;
    union record *rp;
    size_t seg_no;
    unsigned char *ptr;
    struct rt_ant *ant;
    size_t i;

    /* must be double for import and export */
    double v[ELEMENTS_PER_VECT];
    double *vp;

    if (dbip) RT_CK_DBI(dbip);

    BU_CK_EXTERNAL(ep);
    rp = (union record *)ep->ext_buf;
    /* Check record type */
    if (rp->u_id != DBID_ANNOTATION) {
       bu_log("rt_annotation_import4: defective record\n");
       return -1;
    }

    if (bu_debug&BU_DEBUG_MEM_CHECK) {
       bu_log("Barrier check at start of annotation_import4():\n");
       bu_mem_barriercheck();
    }

    RT_CK_DB_INTERNAL(ip);
    ip->idb_major_type = DB5_MAJORTYPE_BRLCAD;
    ip->idb_type = ID_ANNOTATION;
    ip->idb_meth = &OBJ[ID_ANNOTATION];
    BU_ALLOC(ip->idb_ptr, struct rt_annotation_internal);

    annotation_ip = (struct rt_annotation_internal *)ip->idb_ptr;
    annotation_ip->magic = RT_ANNOTATION_INTERNAL_MAGIC;

    /* Warning: type conversion */
    if (mat == NULL) mat = bn_mat_identity;

    bu_cv_ntohd((unsigned char *)v, rp->ano.ant_V, ELEMENTS_PER_VECT);
    MAT4X3PNT(annotation_ip->V, mat, v);

    annotation_ip->vert_count = ntohl(*(uint32_t *)rp->ano.ant_vert_count);
    annotation_ip->ant.count = ntohl(*(uint32_t *)rp->ano.ant_count);
```

```
    ptr = (unsigned char *)rp;
    ptr += sizeof(struct annotation_rec);
    if (annotation_ip->vert_count) {
        annotation_ip->verts = (point2d_t *)bu_calloc(annotation_ip->vert_count,
sizeof(point2d_t), "annotation_ip->vert");
        vp = (double *)bu_calloc(annotation_ip->vert_count, sizeof(double)*ELEMENTS_PER_VECT2D,
"vp");
        bu_cv_ntohd((unsigned char *)vp, ptr, annotation_ip->vert_count*ELEMENTS_PER_VECT2D);

        /* convert double to fastf_t */
        for (i=0; i<annotation_ip->vert_count; i++) {
            annotation_ip->verts[i][X] = vp[(i*ELEMENTS_PER_VECT2D)+0];
            annotation_ip->verts[i][Y] = vp[(i*ELEMENTS_PER_VECT2D)+1];
        }

        bu_free(vp, "vp");
        ptr += 16 * annotation_ip->vert_count;
    }


    if (annotation_ip->ant.count)
        annotation_ip->ant.segments = (void **)bu_calloc(annotation_ip->ant.count, sizeof(void
*), "segs");
    else
        annotation_ip->ant.segments = (void **)NULL;
    for (seg_no=0; seg_no < annotation_ip->ant.count; seg_no++) {
        uint32_t magic;
        struct line_seg *lsg;
        struct carc_seg *csg;
        struct nurb_seg *nsg;
        struct bezier_seg *bsg;
        struct txt_seg *tsg;
        /* must be double for import and export */
        double scan;
        double *scanp;


        magic = ntohl(*(uint32_t *)ptr);
        ptr += SIZEOF_NETWORK_LONG;
        switch (magic) {
            case CURVE_LSEG_MAGIC:
                BU_ALLOC(lsg, struct line_seg);
                lsg->magic = magic;
                lsg->start = ntohl(*(uint32_t *)ptr);
                ptr += SIZEOF_NETWORK_LONG;
                lsg->end = ntohl(*(uint32_t *)ptr);
                ptr += SIZEOF_NETWORK_LONG;
                annotation_ip->ant.segments[seg_no] = (void *)lsg;
                break;
```

```
                case ANN_TSEG_MAGIC:
                BU_ALLOC(tsg, struct txt_seg);
                tsg->magic = magic;
                tsg->ref_pt = ntohl(*(uint32_t *)ptr);
                ptr += SIZEOF_NETWORK_LONG;
                tsg->pos_rel_pt = ntohl(*(uint32_t *)ptr);
                ptr += SIZEOF_NETWORK_LONG;
                tsg->label.vls_offset =  ntohl(*(uint32_t *)ptr);
                ptr += SIZEOF_NETWORK_LONG;
                tsg->label.vls_len = ntohl(*(uint32_t *)ptr);
                ptr += SIZEOF_NETWORK_LONG;
                tsg->label.vls_max = ntohl(*(uint32_t *)ptr);
                ptr += SIZEOF_NETWORK_LONG;
                tsg->label.vls_str = (char *)bu_calloc(tsg->label.vls_len + 1, sizeof(char),
"Annotation label");
        bu_strlcpy(tsg->label.vls_str, ptr, tsg->label.vls_len + 1);
                annotation_ip->ant.segments[seg_no] = (void *)tsg;
                break;
            case CURVE_CARC_MAGIC:
                BU_ALLOC(csg, struct carc_seg);
                csg->magic = magic;
                csg->start = ntohl(*(uint32_t *)ptr);
                ptr += SIZEOF_NETWORK_LONG;
                csg->end = ntohl(*(uint32_t *)ptr);
                ptr += SIZEOF_NETWORK_LONG;
                csg->orientation = ntohl(*(uint32_t *)ptr);
                ptr += SIZEOF_NETWORK_LONG;
                csg->center_is_left = ntohl(*(uint32_t *)ptr);
                ptr += SIZEOF_NETWORK_LONG;
                bu_cv_ntohd((unsigned char *)&scan, ptr, 1);
                csg->radius = scan; /* convert double to fastf_t */
                ptr += SIZEOF_NETWORK_DOUBLE;
                annotation_ip->ant.segments[seg_no] = (void *)csg;
                break;
            case CURVE_NURB_MAGIC:
                BU_ALLOC(nsg, struct nurb_seg);
                nsg->magic = magic;
                nsg->order = ntohl(*(uint32_t *)ptr);
                ptr += SIZEOF_NETWORK_LONG;
                nsg->pt_type = ntohl(*(uint32_t *)ptr);
                ptr += SIZEOF_NETWORK_LONG;
                nsg->k.k_size = ntohl(*(uint32_t *)ptr);
                ptr += SIZEOF_NETWORK_LONG;

                nsg->k.knots = (fastf_t *)bu_malloc(nsg->k.k_size * sizeof(fastf_t),
"nsg->k.knots");
                scanp = (double *)bu_malloc(nsg->k.k_size * sizeof(double), "scanp");
                bu_cv_ntohd((unsigned char *)scanp, ptr, nsg->k.k_size);

                /* convert double to fastf_t */
```

```
                for (i=0; i<(size_t)nsg->k.k_size; i++) {
                    nsg->k.knots[i] = scanp[i];
                }
                bu_free(scanp, "scanp");

                ptr += SIZEOF_NETWORK_DOUBLE * nsg->k.k_size;
                nsg->c_size = ntohl(*(uint32_t *)ptr);
                ptr += SIZEOF_NETWORK_LONG;
                nsg->ctl_points = (int *)bu_malloc(nsg->c_size * sizeof(int),
"nsg->ctl_points");
                for (i=0; i<(size_t)nsg->c_size; i++) {
                    nsg->ctl_points[i] = ntohl(*(uint32_t *)ptr);
                    ptr += SIZEOF_NETWORK_LONG;
                }
                if (RT_NURB_IS_PT_RATIONAL(nsg->pt_type)) {
                    nsg->weights = (fastf_t *)bu_malloc(nsg->c_size * sizeof(fastf_t),
"nsg->weights");
                    scanp = (double *)bu_malloc(nsg->c_size * sizeof(double), "scanp");
                    bu_cv_ntohd((unsigned char *)scanp, ptr, nsg->c_size);

                    /* convert double to fastf_t */
                    for (i=0; i<(size_t)nsg->k.k_size; i++) {
                       nsg->weights[i] = scanp[i];
                    }
                    bu_free(scanp, "scanp");

                    ptr += SIZEOF_NETWORK_DOUBLE * nsg->c_size;
                } else {
                    nsg->weights = (fastf_t *)NULL;
                }
                annotation_ip->ant.segments[seg_no] = (void *)nsg;
                break;
            case CURVE_BEZIER_MAGIC:
                BU_ALLOC(bsg, struct bezier_seg);
                bsg->magic = magic;
                bsg->degree = ntohl(*(uint32_t *)ptr);
                ptr += SIZEOF_NETWORK_LONG;
                bsg->ctl_points = (int *)bu_calloc(bsg->degree + 1, sizeof(int),
"bsg->ctl_points");
                for (i=0; i<=(size_t)bsg->degree; i++) {
                    bsg->ctl_points[i] = ntohl(*(uint32_t *)ptr);
                    ptr += SIZEOF_NETWORK_LONG;
                }
                annotation_ip->ant.segments[seg_no] = (void *)bsg;
                break;
            default:
                bu_bomb("rt_annotation_import4: ERROR: unrecognized segment type!\n");
                break;
        }
    }
```

```
    ant = &annotation_ip->ant;

    if (ant->count)
        ant->reverse = (int *)bu_calloc(ant->count, sizeof(int), "ant->reverse");
    for (i=0; i<ant->count; i++) {
        ant->reverse[i] = ntohl(*(uint32_t *)ptr);
        ptr += SIZEOF_NETWORK_LONG;
    }

    if (bu_debug&BU_DEBUG_MEM_CHECK) {
        bu_log("Barrier check at end of annotation_import4():\n");
        bu_mem_barriercheck();
    }

    return 0;                /* OK */
}
```