

```

/* link_projection.h
 *
 * This file provides the data format in which the UI passes
 * link projections to the kernel. All typedefs begin with
 * "KLP" ("Kernel Link Projection") to avoid name conflicts
 * with the UI's private link projection data structure.
 */

#ifndef _link_projection_
#define _link_projection_

typedef struct KLPCrossing      KLPCrossing;
typedef struct KLPProjection    KLPProjection;

/*
 * The KLPStrandType and KLPDirectionType enums are used to index
 * array entieres, so their values must be 0 and 1. (But the code
 * does not rely on which has value 0 and which has value 1.)
 *
 * JRW 2000/11/12   Use fake "typedef enums" instead of real ones,
 * for the reasons explained at the top of kernel_typedefs.h.
 */

/*
 * If you view a crossing (from above) so that the strands go in the
 * direction of the postive x- and y-axes, then the strand going in
 * the x-direction is the KLPStrandX, and the strand going in the
 * y-direction is the KLPStrandY. Note that this definition does not
 * depend on which is the overstrand and which is the understrand.
 *
 *
 *           KLPStrandY
 *             ^
 *            |
 * -----+----> KLPStrandX
 *            |
 *             v
 */
typedef int KLPStrandType;
enum
{
    KLPStrandX = 0,
    KLPStrandY,
    KLPStrandUnknown
};

/*
 * The backward and forward directions are what you would expect.
 *
 *          KLPBackward   ----->     KLPForward
 */
typedef int KLPDirectionType;
enum
{
    KLPBackward = 0,
    KLPForward,
    KLPDirectionUnknown
};

/*
 * A crossing is either a clockwise (CL) or counterclockwise (CCL)
 * half twist.
 *
 *
 *         \       /
        ____\____/____
        //      \\
 *
 *           KLPHalfTwistCL
 *
 *
 *         \       /
        ____\____/____
        //      \\
 *
 *           KLPHalfTwistCCL
 */
typedef int KLPCrossingType;
enum
{

```

```

    KLPHalfTwistCL,
    KLPHalfTwistCCL,
    KLPCrossingTypeUnknown
};

/*
 * A link projection is essentially an array of crossings.
 */
struct KLPPProjection
{
    /*
     * How many crossings are there?
     */
    int num_crossings;

    /*
     * How many free loops (i.e. link components with no crossings) are
     * there? For a hyperbolic link, the number of free loops must be 0.
     */
    int num_free_loops;

    /*
     * How many link components (including the free loops) are there?
     */
    int num_components;

    /*
     * Here's a pointer to the array of crossings.
     */
    KLPCrossing *crossings;
};

/*
 * Each crossing has pointers to its four neighbors,
 * along with information about its own handedness.
 */
struct KLPCrossing
{
    /*
     * The four neighbors are
     *
     *     neighbor[KLPStrandX][KLPBackward]
     *     neighbor[KLPStrandX][KLPForward ]
     *     neighbor[KLPStrandY][KLPBackward]
     *     neighbor[KLPStrandY][KLPForward ]
     *
     * For example, if you follow the x-strand in the backward direction,
     * you'll arrive at neighbor[KLPStrandX][KLPBackward].
     */
    KLPCrossing *neighbor[2][2];

    /*
     * When you arrive at a neighbor, you could arrive at either the
     * x-strand or the y-strand. The strand[][] field says which it is.
     *
     * For example, if you follow the x-strand in the backward direction,
     * you'll arrive at strand[KLPStrandX][KLPBackward].
     */
    KLPStrandType strand[2][2];

    /*
     * The crossing is either a clockwise or counterclockwise half twist.
     */
    KLPCrossingType handedness;

    /*
     * To which component of the link does each strand belong? A link
     * component is given by an integer from 0 to (#components - 1).
     * For example, if component[KLPStrandX] == 0 and
     * component[KLPStrandY] == 2, then the x-strand is part of component
     * number 0 and the y-strand is part of component number 2.
     */
    int component[2];
};

```

};

#endif