This document was written to accompany CODE version 4.1. The information contained in this document is subject to change without notice. Slight variations may occur on computers from varying manufacturers. Every effort has been made to supply complete and accurate information. However, Cimetrix Inc. makes no warranty of any kind with regard to this manual, including, but not limited to the implied warranties of merchantability and fitness for a particular purpose. Cimetrix shall not be liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this manual.

Copyright © 1990 - 1999 Cimetrix Inc.

This document is the property of Cimetrix Inc. and is protected by copyright. The information contained herein is proprietary and may not be copied, transferred, or disclosed to others without prior written authorization from Cimetrix Inc., Salt Lake City, Utah 84047-3757.

Portions of the CODE System are protected by US Patent 4,831,549.

print date: 04/27/99 4:25 PM

Trademark Acknowledgments

Terms mentioned in this document that are known to be trademarks are listed below. Cimetrix Inc. cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark.

Cimetrix is a registered trademark of Cimetrix, Inc.
The Cimetrix logo, CODE, CIMServer, CIMulation, CIMControl, CIMTools, CODE API, CIMCal, CIMApp and CIMTeach are trademarks of Cimetrix Inc.
80386, 80486, and i486 are trademarks of Intel.
PostScript is a registered trademark of Adobe Systems, Inc.
Venture Com
PMAC
MEI
All other products are trademarks or registered trademarks of their respective companies.

Part Number (CODE-API-4.1.0 NT V2 05/99)
C Function Prototypes

Interference Checking

long CxCheckIntx(CxServer Server, long intx_mind, CxNodeId node1, long branch1, CxNodeId node2, long branch2, long override_flag, long block_curves, long block_triads, long tool_check, long self_check, double rs_mind, double rs_near, long *intx, double *mind, CxNodeId *res_node1, CxNodeId *res_node2)

long CxGetIntxType(CxServer Server, CxNodeId node)

long CxSetIntxType(CxServer Server, CxNodeId node, long children, long setting)

Rendering

long CxChangeColor(CxServer Server, CxNodeId node, double r, double g, double b)

long CxChangeStructColor(CxServer Server, CxNodeId node, double r, double g, double b)

long CxDullnessLevel(CxServer Server, CxNodeId node, long level)

long CxFrameSwitch(CxServer Server, CxNodeId node, long onoff)

long CxGetIntxType(CxServer Server, CxNodeId node)

long CxGetColor(CxServer Server, CxNodeId node, double *r, double *g, double *b)

long CxGetDefaultView(CxServer Server, long *type)

long CxGetDullnessLevel(CxServer Server, CxNodeId node, long *level)

long CxGetFrameSwitch(CxServer Server, CxNodeId node, long *onoff)

long CxGetReflectivity(CxServer Server, CxNodeId node, double *r, double *g, double *b)

long CxGetTransparency(CxServer Server, CxNodeId node, long *level)

long CxGetTriadSize(CxServer Server, CxNodeId node, double *size)

long CxGetViewSwitch(CxServer Server, CxNodeId node, long *onoff)

long CxGetViewType(CxServer Server, CxNodeId node, long *type)

long CxGraphicsSwitch(CxServer Server, long onoff)

long CxReflectivity(CxServer Server, CxNodeId node, double r, double g, double b)

long CxSetDefaultView(CxServer Server, long type)

long CxSetTriadSize(CxServer server, CxNodeId node, double size)

long CxStructDullnessLevel(CxServer Server, CxNodeId node, long level)
long CxStructFrameSwitch(CxServer Server, CxNodeId node, long onoff)
long CxStructReflectivity(CxServer Server, CxNodeId node, double r, double g, double b)
long CxStructTransparency(CxServer Server, CxNodeId node, long level)
long CxStructViewSwitch(CxServer Server, CxNodeId ele, long onoff)
long CxStructViewType(CxServer Server, CxNodeId ele, long type)
long CxTransparency(CxServer Server, CxNodeId ele, long level)
long CxViewSwitch(CxServer Server, CxNodeId ele, long onoff)
long CxViewType(CxServer Server, CxNodeId ele, long type)

Tool Path Display
long CxAddTrail(CxServer Server, CxNodeId node)
long CxCalcFrameSwitch(CxServer Server, CxNodeId node long onoff, long freq)
long CxCalcTrailSwitch(CxServer Server, CxNodeId node, long onoff)
long CxClearTrail(CxServer Server, CxNodeId node)
long CxDeleteTrail(CxServer Server, CxNodeId node)
long CxGetCalcFrameSwitch(CxServer Server, CxNodeId node, long *onoff, long *freq)
long CxGetCalcTrailSwitch(CxServer Server, CxNodeId node, long *onoff)
long CxGetMoveDraw(CxServer Server, CxNodeId node, long *movedraw)
long CxGetNumTrailMoves(CxServer Server, CxNodeId node, long *num)
long CxGetRefNode(CxServer Server, CxNodeId node, CxNodeId *ref_node)
long CxGetTrailColor(CxServer Server, CxNodeId node, double *r, double *g, double *b)
long CxGetTrailViewSwitch(CxServer Server, CxNodeId node, long *lonoff, long *fronoff)
long CxSetMoveDraw(CxServer Server, CxNodeId node, long movedraw)
long CxSetNumTrailMoves(CxServer Server, CxNodeId node, long num)
long CxSetRefNode(CxServer Server, CxNodeId node, CxNodeId ref_node)
long CxSetTrailColor(CxServer Server, CxNodeId node, double r, double g, double b)
long CxSetTrailViewSwitch(CxServer Server, CxNodeId node, long lonoff, long fronoff)

Matrix
long CxCompareMat(CxMatrix mat1, CxMatrix mat2, CxMatrix rel_mat, double position_tol, double screw_tol, long comp_type)
void CxCross(CxVector vec_x, CxVector vec_y, CxVector vec_z)
double CxDistan(CxVector vec1, CxVector vec2)
double CxDotp(double factor, CxVector vec1, CxVector vec2)
double CxDots(CxVector vec1, CxVector vec2)
void Euler(CxMatrix mat, CxVector angles)
void CxFindtr(CxMatrix mat, CxVector trans_vec)
void CxXyzang(CxMatrix mat, CxVector angles)
long CxXyzmat(char axes[4], CxVector angles, CxVector trans_vec, CxMatrix mat)

**PMAC- Specific Functions**
long CxConvertGatherData(char *buffer, char *format, double *data, long *data_size, long *err_no)
long CxPmacGatherInit(CxController cntrl, long num_sig, long *signals, long period, long mode, long *num_sample, long *buff_size, char *format)
long CxPmacGatherStart(CxController cntrl, long trigger)
long CxPmacGatherStop(CxController cntrl)
long CxPmacGatherUpload(CxController cntrl, long start_sample, long num_sample, long *actual_sample, long buff_size, char *buffer)
long CxPmacGetIvar(CxController cntrl, long ivar, double *value)
long CxPmacSetIvar(CxController cntrl, long ivar, double value)

**MEI DSP – Specific Functions**
long CxDspDisableCamming (CxMechanism mech, int cam_axis)
long CxDspDisableGearing (CxMechanism mech, int slave)
long CxDspDisablePredictiveLimits (CxMechanism mech, int axis_index)
long CxDspEnableCamming (CxMechanism mech, int master_axis, int cam_axis, DspCammingSource mode)
long CxDspEnableGearing (CxMechanism mech, int master, int slave, double ratio, DspGearingSource mode)
long CxDspEnablePredictiveLimits (CxMechanism mech, int axis_indent, double margin)
long CxDspGetESTopRate (CxMechanism mech, int axis_indent, double *e_stop_rate)
long CxDspGetNegativeSWLimit (CxMechanism mech, int axis_indent, double *limit_pos)
long CxDspGetNegativeSWLimitPolicy (CxMechanism mech, int axis_index, DspLimitPolicy *policy)
long CxDspGetPositiveSWLimit (CxMechanism mech, int axis_index, double *limit_pos)
long CxDspGetPositiveSWLimitPolicy (CxMechanism mech, int axis_index, DspLimitPolicy *policy)
long CxDspGetShortMoveMode (CxMechanism mech, CxShortMoveMode *mode, double *max_jerk, double *min_time)
long CxDspGetStopRate (CxMechanism mech, int axis_index, double *stop_rate)
long CxDspQueuedDisableCamming (CxMechanism mech, int cam_axis)
long CxDspQueuedDisableGearing (CxMechanism mech, int slave)
MEI XMP – Specific Functions

```c
long CxDspQueuedDisablePredictiveLimits (CxMechanism mech, int axis_index)
long CxDspQueuedEnableCamming (CxMechanism mech, int master_axis, int cam_axis, DspCammingSource mode)
long CxDspQueuedEnableGearing (CxMechanism mech, int master, int slave, double ratio, DspGearingSource mode)
long (CxMechanism mech, int axis_index, double margin)
long CxDspQueuedSetNegativeSWLimitPolicy (CxMechanism mech, int axis_index, DspLimitPolicy *policy)
long CxDspQueuedSetPositiveSWLimitPolicy (CxMechanism mech, int axis_index, DspLimitPolicy *policy)
long CxDspQueuedSetShortMoveMode (CxMechanism mech, CxShortMoveMode mode, double max_jerk, double min_time)
long CxDspSetESStopRate (CxMechanism mech, int axis_index, double e_stop_rate)
long CxDspSetNegativeSWLimit (CxMechanism mech, int axis_index, double limit_pos)
long CxDspSetNegativeSWLimitPolicy (CxMechanism mech, int axis_index, DspLimitPolicy *policy)
long CxDspSetPositiveSWLimit (CxMechanism mech, int axis_index, double limit_pos)
long CxDspSetPositiveSWLimitPolicy (CxMechanism mech, int axis_index, DspLimitPolicy policy)
long CxDspSetShortMoveMode (CxMechanism mech, CxShortMoveMode mode, double max_jerk, double min_time)
long CxDspSetStopRate (CxMechanism mech, int axis_index, double stop_rate)
long CxGetDacLimit (CxMechanism mech, int axis_index, double *voltage_limit)
long CxSetDacLimit (CxMechanism mech, int axis_index, double voltage_limit)
```
long CxXmpQueuedSetAxisConfig (CxMechanism mech, long joint, long mask, MEIPositionConfig *config, long configSize)

long CxXmpSetAxisConfig (Cxmechanism mech, long joint, long mask, MEIPositionConfig *config, long configSize)

long CxXmpSetJerkPercent (CxMechanism mech, double jerk_percent)

long CxXmpSetJointInterpType (CxMechanism mech, long joint_interp_type)

long CxXmpSetRapidJerkPercent (CxMechanism mech, long joint, double jerk_percent)

Anorad IPC Specific Functions

long CxAnoradDisableJoint (CxMechanism mech, long joint)

long CxAnoradDisableDataCollection (CxMechanism mech)

long CxAnoradDisableErrorMap (CxMechanism mech, long map)

long CxAnoradDwell (CxMechanism mech, long milliseconds)

long CxAnoradEnableDataCollection (CxMechanism mech)

long CxAnoradEnableErrorMap (CxMechanism mech, long map, long joint1, long joint2)

long CxAnoradEnableJoint (CxMechanism mech, long joint)

long CxAnoradFlushDataCollection (CxMechanism mech)

long CxAnoradFlushStrobeBuffer (CxMechanism mech)

long CxAnoradFreeErrorMap (CxMechanism mech, long map)

long CxAnoradGetDataCollection (CxMechanism mech, double *values)

long CxAnoradGetDataCollectionLeft (CxMechanism mech, long *value)

long CxAnoradGetErrorStruct (CxMechanism mech, CxAnoradErrorstruct *errStruct)

long CxAnoradGetJointStatus (CxMechanism mech, unsigned long *status)

long CxAnoradGetStrobeLeft (CxMechanism mech, long *value)

long CxAnoradGetTrackValue (CxMechanism mech, long joint, double *value)

long CxAnoradHomeJoint (CxMechanism mech, long joint)

long CxAnoradLoadErrorMap (CxMechanism mech, long map, char *filename)

long CxAnoradSetStrobePoint (CxMechanism mech, long joint, double position)

long CxAnoradSetTrack (CxMechanism mech, long joint, short inrange, short value, short error, short clip, short clip2, short start, float gain)

long CxAnoradSetTrackValue (CxMechanism mech, long joint, short value)

long CxAnoradTrackOff (CxMechanism mech, long joint)

long CxAnoradTrackOn (CxMechanism mech, long joint)
long CxAnoradWaitForDataCollection (CxMechanism mech, double*values, long timeout)

CIMCal Functions

(See the CIMCal Manual).
Interference Checking
**CxCheckIntx**

Checks the intersection and minimum distance

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxCheckIntx (CxServer Server, long intx_mind, CxNodeId node1, long branch1, CxNodeId node2, long branch2, long override_flag, long block_curves, long block_triads, long tool_check, long self_check, double rs_mind, double rs_near, long *intx, double *mind, CxNodeId *res_node1, CxNodeId *res_node2)
```

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>Server ID</td>
</tr>
<tr>
<td>intx_mind</td>
<td>Flag indicating type of check to perform. Valid constants (checks) are:</td>
</tr>
<tr>
<td></td>
<td>CX_INTX_ONLY Only collision checking will be performed.</td>
</tr>
<tr>
<td></td>
<td>CX_INTX_NEAR Nodes within the rs_near sensitivity tolerance will be reported as collisions.</td>
</tr>
<tr>
<td></td>
<td>CX_INTX_MIND Actual minimum distance between nodes will be calculated, unless a collision is detected.</td>
</tr>
<tr>
<td>node1</td>
<td>Node ID for the first node to be checked</td>
</tr>
<tr>
<td>branch1</td>
<td>Flag which determines whether children descending from node1 will be included in the check. If the flag is CX_TRUE, the descendants with intersection checking will be enabled are included in the check. If CX_FALSE, only node1 will be checked.</td>
</tr>
<tr>
<td>node2</td>
<td>Node ID for the second node to be checked</td>
</tr>
<tr>
<td>branch2</td>
<td>Flag which determines whether children descending from node2 will be included in the check. If the flag is CX_TRUE, the descendants with intersection checking enabled are included in the check. If CX_FALSE, only node2 will be checked.</td>
</tr>
<tr>
<td>override_flag</td>
<td>Flag which determines if all nodes selected for checking will be checked, regardless of whether intersection and minimum distance checking is enabled for that node (CX_TRUE) or (CX_FALSE)</td>
</tr>
<tr>
<td>block_curves</td>
<td>Flag which excludes nodes defined as cuves from checking when set to CX_TRUE</td>
</tr>
<tr>
<td>block_triads</td>
<td>A flag which excludes nodes with no GEOMETRY attributes from checking when set to CX_TRUE. This flag is ignored if one or both of the nodes being checked is a triad, and branch is not selected for that node.</td>
</tr>
<tr>
<td>tool_check</td>
<td>Flag, if set to CX_TRUE, will perform a check on the whole robot against the other node or branch. In addition, all of the nodes below the last joint frame of the robot are then checked against the robot nodes as described below under self_check. One of the selected nodes (node1 or node2) must be a robot and the branch for that node must be set to CX_TRUE.</td>
</tr>
<tr>
<td>self_check</td>
<td>Flag, if set to CX_TRUE, will check all the nodes of the robot below the last joint frame which have collision checking enabled against the rest of the robot. The application needs to define only one node for a robot self-check, which must be a robot node. An error will be returned if the node is not a robot and the entire robot branch has not been selected. This check is useful when the robot tool picks up a part which may collide with the inner members of the robot.</td>
</tr>
</tbody>
</table>
rs_mind  Sensitivity tolerance for the true minimum distance checking. Nodes outside this tolerance will not be included in the check.
rs_near  Sensitivity distance for near-miss checking. If nodes are within this tolerance, they are treated as collisions.
intx  Flag which returns the value CX_INTX if an intersection occurs; otherwise, no intersection occurs.
mind  Calculated minimum distance if CX_INTX_MIND check was enabled.
res_node1  One of the two nodes involved in the collision or for which the minimum distance is determined.
res_node2  One of the two nodes involved in the collision or for which the minimum distance is determined.

DESCRIPTION
This function is the interface function allowing users to determine whether or not nodes in a workcell will collide, or to calculate the minimum distance between any two nodes or groups of nodes.

RETURN VALUES
This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS
If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
<tr>
<td>CX_NODE_NOT_SELECTED</td>
<td>One of the nodes entered is not a valid node.</td>
</tr>
<tr>
<td>CX_INVALID_INTX_CHECK_TYPE</td>
<td>Check type entered is not CX_INTX_ONLY, CX_INTX_NEAR, or CX_INTX_MIND</td>
</tr>
<tr>
<td>CX_ROBOT_BRANCH_NOT_SELECTED</td>
<td>self_check or tool_check requires a robot branch</td>
</tr>
<tr>
<td>CX_NODE_INTX_OFF</td>
<td>Nodes involved are not turned on for checking</td>
</tr>
<tr>
<td>CX_NO_MEMORY</td>
<td>Memory error in CX_INTX algorithms</td>
</tr>
<tr>
<td>CX_NODE_NOT_ROBOT</td>
<td>self_check or tool_check requires robot branch</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Given node does not exist.</td>
</tr>
</tbody>
</table>
SEE ALSO

CxSetIntxType, CxGetIntxType
**CxGetIntxType**

Determines if intersection and minimum distance checking is enabled

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxGetIntxType (CxServer Server, CxNodeId node)
```

**ARGUMENTS**

- **Server**
  - Server ID
- **node**
  - Node ID

**DESCRIPTION**

This function determines whether or not intersection and minimum distance checking is enabled for a node. If a node is disabled for intersection and minimum distance checking the node will not be included in an interference check unless the override flag is set, an interference check is initiated using either the CIMTools Interference Checking dialog box or the `CxCheckIntx` function.

**RETURN VALUES**

This function returns a positive value if intersection checking is enabled, 0 if disabled, or -1 (`CX_ERROR`) if an error occurs.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the `CxGetErrorNumber` function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Given node does not exist.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`CxSetIntxType`, `CxCheckIntx`
**CxSetIntxType**

Enables or disables intersection and minimum distance checking

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxSetIntxType (CxServer Server, CxNodeId node, long children, long setting)
```

**ARGUMENTS**

- **Server**  
  Server ID
- **node**  
  Node ID
- **children**  
  Flag which enables (CX_TRUE) or disables (CX_FALSE) intersection and minimum distance checking for all children descending from node
- **setting**  
  Flag to enable (CX_ON) or disable (CX_OFF) intersection and minimum distance checking

**DESCRIPTION**

This function enables or disables intersection and minimum distance checking on a node or group of nodes. If a node is disabled for intersection and minimum distance checking, then when an interference check is initiated using either the CIMTools Interference Checking dialog box or the CxCheckIntx function that node will not be included in the check unless the override flag is set.

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Given node does not exist.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

CxGetIntxType, CxCheckIntx
Rendering
**ListeningColor**

Changes the color of a node having the GEOMETRY attribute

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxChangeColor(CxServer Server, CxNodeId node, double r, double g, double b)
```

**ARGUMENTS**

- `Server`  
  Server ID
- `node`  
  Node ID
- `r`  
  Red component (range: 0.0 < r < 1.0)
- `g`  
  Green component (range: 0.0 < g < 1.0)
- `b`  
  Blue Component (range: 0.0 < b < 1.0)

**DESCRIPTION**

This function changes the color of the given node. The node must have the GEOMETRY attribute. If the component value specified does not fall within the range, it will be set to the nearest boundary value (i.e., value < 0 will be set to 0.0 and value > 1 will be set to 1.0).

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>Node does not have the GEOMETRY attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following program changes the color of the table.
#include <code/robpac.h>

void main(void)
{
    CxServer    Server;
    CxMechanism mech;
    CxNodeId   table;

    Server = CxOpenServer ("Testing", CX_SMEM, 0);

    CxGetNamedNodeId (Server, "table", &table);
    if (CxChangeColor (Server, table, 0.5, 0.5, 0.5) == CX_ERROR)
        printf("Error in changing the color of the table \n");

    CxRobpacExit();
}

SEE ALSO

CxChangeStructColor, CxGetColor
**CxChangeStructColor**

Changes the color of a branch of nodes

**SYNOPSIS**

```
#include <code/robpac.h>
long CxChangeStructColor(CxServer Server, CxNodeId node, double r,
                        double g, double b)
```

**ARGUMENTS**

- **Server**: Server ID
- **node**: Node ID
- **r**: Red component (range: 0.0 ≤ r ≤ 1.0)
- **g**: Green component (range: 0.0 ≤ g ≤ 1.0)
- **b**: Blue component (range: 0.0 ≤ b ≤ 1.0)

**DESCRIPTION**

This function is used to change the color of the given node and the whole branch of its child nodes. The color of all nodes within the branch that have the GEOMETRY attribute will be changed. If the component value specified does not fall within the range, it will be set to the nearest boundary value (i.e., value < 0 will be set to 0.0 and value > 1 will be set to 1.0).

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

CxChangeColor, CxGetColor
**CxDullnessLevel**

Changes the dullness level of a node having the GEOMETRY attribute

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxDullnessLevel(CxServer Server, CxNodeId node, long level)
```

**ARGUMENTS**

- **Server**  
  Server ID
- **node**  
  Node ID
- **onoff**  
  Switch CX_ON or CX_OFF

**DESCRIPTION**

This function changes the dullness level of the given node. The node must have the GEOMETRY attribute. If the dullness level specified does not fall within the range, it will be set to the nearest boundary value (i.e., value < 1 will be set to 1 and value > 400 will be set to 400).

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>Node does not have the GEOMETRY attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

CxStructDullnessLevel, CxGetDullnessLevel
CxFrameSwitch

Turns a node’s triad on or off

SYNOPSIS

#include <code/robpac.h>
long CxFrameSwitch(CxServer Server, CxNodeId node, long onoff)

ARGUMENTS

Server Server ID
node Node ID
onoff Switch CX_ON or CX_OFF

DESCRIPTION

This function turns a node’s frame view on or off, or, in other words, this function determines whether the triad
representing the node (element) is displayed in the CIMServer graphics window.

RETURN VALUES

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber
function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE.Receive_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_NODE.IS_CUT.OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

SEE ALSO

CxStructFrameSwitch, CxGetFrameSwitch
**Synopsis**

```c
#include <code/robpac.h>
long CxGetColor(CxServer Server, CxNodeId node, double *r, double *g, double *b)
```

**Arguments**

- `Server` Server ID
- `node` Node ID
- `r` Red component (range: `0.0 < r < 1.0`)
- `g` Green component (range: `0.0 < g < 1.0`)
- `b` Blue component (range: `0.0 < b < 1.0`)

**Description**

This function gets the color of the given node. The node must have the `GEOMETRY` attribute.

**Return Values**

This function returns `0` if successful; otherwise, `-1` (`CX_ERROR`) is returned.

**Errors**

If the function returns an error condition, the error code can be obtained by using the `CxGetErrorNumber` function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>Node does not have the <code>GEOMETRY</code> attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

**See Also**

`CxChangeColor`, `CxChangeStructColor`
CxGetDefaultView

Gets the view type that all nodes with normal view type will have

SYNOPSIS

#include <code/robpac.h>
long CxGetDefaultView(CxServer Server, long *type)

ARGUMENTS

Server Server ID

type Default view type (possible types listed below)

DESCRIPTION

This function gets the default view type for all geometry nodes in the server. Each geometry node will be drawn with this setting unless otherwise specified (see CxViewType.) The possible default view types are listed below.

CX_CLEAR Transparent
CX_CLEAR_EDGED Transparent edged
CX_FILLED Filled
CX_FILLED_EDGED Filled edged
CX_SHADED Flat shaded
CX_SHADED_EDGED Flat shaded, edged
CX_SMOOTH Smooth shaded
CX_SMOOTH_EDGED Smooth shaded, edged
CX_WIRE Wire frame

RETURN VALUES

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
</tbody>
</table>

SEE ALSO

CxSetDefaultView, CxViewType, CxGetViewType
**CxGetDullnessLevel**

Gets the dullness level of a node

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxGetDullnessLevel(CxServer Server, CxNodeId node, long *level)
```

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Server</th>
<th>Server ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>node</td>
<td>Node ID</td>
</tr>
<tr>
<td>level</td>
<td>Dullness level</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

This function gets the dullness level of the given node. The node must have the `GEOMETRY` attribute.

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the `CxGetErrorNumber` function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>Node does not have the <code>GEOMETRY</code> attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`CxDullnessLevel`, `CxStructDullnessLevel`
CxGetFrameSwitch
Determines if a node’s triad is on or off

SYNOPSIS
#include <code/robpac.h>
long CxGetFrameSwitch(CxServer Server, CxNodeId node, long *onoff)

ARGUMENTS
Server Server ID
node Node ID
level Dullness level

DESCRIPTION
This function determines whether a node's frame view (triad) is on or off. If onoff is returned as CX_ON, the triad representing the node (element) is currently displayed in the CIMServer graphics window. If it is returned as CX_OFF, the triad is not being currently displayed.

RETURN VALUES
This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS
If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

SEE ALSO
CxFrameSwitch, CxStructFrameSwitch
**CxGetReflectivity**

Gets a node’s specular reflectivity coefficients

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxGetReflectivity(CxServer Server, CxNodeId node, double *r, double *g, double *b)
```

**ARGUMENTS**

- **Server**: Server ID
- **node**: Node ID
- **r**: Red component (range: 0.0 < r < 1.0)
- **g**: Green component (range: 0.0 < g < 1.0)
- **b**: Blue component (range: 0.0 < b < 1.0)

**DESCRIPTION**

This function determines the given node’s specular reflectivity coefficients. The node must have the GEOMETRY attribute.

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>Node does not have the GEOMETRY attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

CxReflectivity, CxStructReflectivity
**CxGetTransparency**

Gets the transparency level of a node

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxGetTransparency(CxServer Server, CxNodeId node, long *level)
```

**ARGUMENTS**

- `Server` : Server ID
- `node` : Node ID
- `level` : Transparency level (range $0 \leq \text{level} \leq 16$ where 16 is maximum transparency)

**DESCRIPTION**

This function gets the transparency level of the given node. The node must have the `GEOMETRY` attribute.

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (`CX_ERROR`) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the `CxGetErrorNumber` function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>Node does not have the <code>GEOMETRY</code> attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

CxTransparency, CxStructTransparency
**CxGetTriadSize**

Gets the size of the triad for a specified node

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxGetTriadSize (CxServer Server, CxNodeId node, double *size)
```

**ARGUMENTS**

- **Server**: Server ID
- **node**: Node ID
- **size**: Size of the triad (default is .75)

**DESCRIPTION**

This API gets the size of the triad for a specified node.

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>An error in sending the message</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>An error in receiving the message</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>The named node does not exist</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The given node has been cut out</td>
</tr>
<tr>
<td>CX_AMBIGUOUS_SEARCH_PATH</td>
<td>More than one path contains the information provided</td>
</tr>
<tr>
<td>CX_NODE_NAME_AMBIGUOUS</td>
<td>More than one node contains the given name</td>
</tr>
</tbody>
</table>

**EXAMPLE**

```c

CxServer pServer;
CxNodeId myNode;
double dSize;

pServer = CxOpenServer( "myServerName", CX_LOCAL_PROTOCOL, 0 );
CxGetNamedNodeId( pServer, "world", &myNode );

//Do some stuff here

CxGetTriadSize( pServer, myNode, &dSize );
CxSetTriadSize( pServer, myNode, 10.0 );

//Do some more stuff

//Reset the triad size to the original size
CxSetTriadSize( pServer, myNode, dSize );
```
SEE ALSO

CxSetTriadSize
**CxGetViewSwitch**

Determines if a node’s view is on or off

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxGetViewSwitch(CxServer Server, CxNodeId node, long *onoff)
```

**ARGUMENTS**

- **Server**  
  Server ID
- **node**  
  Node ID
- **onoff**  
  Switch (CX_ON or CX_OFF)

**DESCRIPTION**

This function determines if the given node's view is on or off. The node must have the GEOMETRY attribute. If onoff is returned as CX_ON, the node (element) is currently displayed in the CIMServer graphics window. If it is returned as CX_OFF, the node is not being currently displayed.

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>Node does not have the GEOMETRY attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

CxViewSwitch, CxStructViewSwitch
**CxGetViewType**

Gets a node’s view type

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxGetViewType(CxServer Server, CxNodeId node, long *type)
```

**ARGUMENTS**

- **Server**  Server ID
- **node**  Node Id
- **type**  View type (see list in **DESCRIPTION**)

**DESCRIPTION**

This function gets the given node's view type. Listed below are the possible view types and their descriptions:

<table>
<thead>
<tr>
<th>View Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_CLEAR</td>
<td>Transparent</td>
</tr>
<tr>
<td>CX_CLEAR_EDGED</td>
<td>Transparent edged</td>
</tr>
<tr>
<td>CX_FILLED</td>
<td>Filled</td>
</tr>
<tr>
<td>CX_FILLED_EDGED</td>
<td>Filled edged</td>
</tr>
<tr>
<td>CX_NORMAL</td>
<td>View type set by CxSetDefaultView</td>
</tr>
<tr>
<td>CX_SHADED</td>
<td>Flat shaded</td>
</tr>
<tr>
<td>CX_SHADED_EDGED</td>
<td>Flat shaded, edged</td>
</tr>
<tr>
<td>CX_SMOOTH</td>
<td>Smooth shaded</td>
</tr>
<tr>
<td>CX_SMOOTH_EDGED</td>
<td>Smooth shaded, edged</td>
</tr>
<tr>
<td>CX_WIRE</td>
<td>Wire frame</td>
</tr>
</tbody>
</table>

The node must have the `GEOMETRY` attribute.

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>Error Codes</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>Node does not have the GEOMETRY attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

SEE ALSO

CxViewType, CxStructViewType, CxGetDefaultView, CxSetDefaultView
**CxGraphicsSwitch**

Turning graphics redraw on or off

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxGraphicsSwitch(CxServer Server, long onoff)
```

**ARGUMENTS**

- `Server`: Server ID
- `onoff`: Redraw (CX_ON or CX_OFF)

**DESCRIPTION**

This function toggles the redraw capabilities of a graphics server. If the redraw is off, commands continue to be executed by the graphics server, but any changes made will not be shown in the graphical server. When the redraw is turned back on, the graphical server is updated with all of the changes made while the redraw was not on. This may be useful if updating the graphics server would take a long time (i.e.: drawing a curve with many curve segments) because the user application can toggle the switch, draw the curve, then turn the switch back on. The curve will then appear without the slow redraw.

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
</tbody>
</table>
**CxReflectivity**

Changes a node’s specular reflectivity coefficients

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxReflectivity(CxServer Server, CxNodeId node, double r, double g, double b)
```

**ARGUMENTS**

- `Server`: Server ID
- `node`: Node ID
- `r`: Red component (range: 0.0 < r < 1.0)
- `g`: Green component (range: 0.0 < g < 1.0)
- `b`: Blue component (range: 0.0 < b < 1.0)

**DESCRIPTION**

This function changes the specular reflectivity coefficients of the given node. The node must have the GEOMETRY attribute; otherwise, an error is returned. If the component value specified does not fall within the range, it will be set to the nearest boundary value (i.e., value < 0 will be set to 0.0, and value > 1 will be set to 1.0).

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>Node does not have the GEOMETRY attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

CxStructReflectivity, CxGetReflectivity
CxSetDefaultView
Sets the view that all nodes with normal view type will have

SYNOPSIS
#include <code/robpac.h>
long CxGetDefaultView(CxServer Server, long type)

ARGUMENTS
Server Server ID
type Default view type (possible types listed below)

DESCRIPTION
This function sets the default view type for all geometry nodes in the server. Each geometry node will be drawn
with this setting unless otherwise specified (see CxViewType). The possible default view types are listed below.
CX_CLEAR Transparent
CX_CLEAR_EDGED Transparent edged
CX_FILLED Filled
CX_FILLED_EDGED Filled edged
CX_SHADED Flat shaded
CX_SHADED_EDGED Flat shaded, edged
CX_SMOOTH Smooth, shaded
CX_SMOOTH_EDGED Smooth shaded, edged
CX_WIRE Wire frame

RETURN VALUES
This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS
If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber
function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
</tbody>
</table>

SEE ALSO
CxGetDefaultView, CxViewType, CxGetViewType
**CxSetTriadSize**

Sets the size of the triad for a specified node

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxSetTriadSize (CxServer Server, CxNodeId node, double size)
```

**ARGUMENTS**

- **Server**: Server ID
- **node**: Node ID
- **size**: Size of the triad (default is .75)

**DESCRIPTION**

This API sets the displays the size of the triad for a specified node.

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>An error in sending the message</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>An error in receiving the message</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>The named node does not exist</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The given node has been cut out</td>
</tr>
<tr>
<td>CX_AMBIGUOUS_SEARCH_PATH</td>
<td>More than one path contains the information provided</td>
</tr>
<tr>
<td>CX_NODE_NAME_AMBIGUOUS</td>
<td>More than one node contains the given name</td>
</tr>
</tbody>
</table>

**EXAMPLE**

```c

CxServer pServer;
CxNodeId myNode;
double dSize;
pServer = CxOpenServer( "myServerName", CX_LOCAL_PROTOCOL, 0 );
CxGetNamedNodeId( pServer, "world", &myNode );
//Do some stuff here
CxGetTriadSize( pServer, myNode, &dSize );
CxSetTriadSize( pServer, myNode, 10.0 );
//Do some more stuff
//Reset the triad size to the original size
CxSetTriadSize( pServer, myNode, dSize );
```
SEE ALSO

CxSetTriadSize
**CxStructDullnessLevel**

Changes the dullness level of a branch of nodes

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxStructDullnessLevel(CxServer Server, CxNodeId node, long level)
```

**ARGUMENTS**

- **Server** Server ID
- **node** Node ID
- **level** Dullness level (1 ≤ level ≤ 400, where 1 is the brightest)

**DESCRIPTION**

This function changes the dullness level of the given node and the branch made up of its child nodes. The dullness level of all nodes within the branch that have the GEOMETRY attribute will be changed. If the dullness level specified does not fall within the range, it will be set to the nearest boundary value (i.e., level < 1 will be set to 1, and level > 400 will be set to 400.)

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE.Receive_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

CxDullnessLevel, CxGetDullnessLevel
CxStructFrameSwitch

Turns triad on or off for a branch of nodes

SYNOPSIS

#include <code/robpac.h>
long CxStructFrameSwitch(CxServer Server, CxNodeId node, long onoff)

ARGUMENTS

Server Server Id
node Node ID
onoff Switch (CX_ON or CX_OFF)

DESCRIPTION

This function is used to turn frame view (triad) on or off for the given node and the branch made up of its child nodes. If set to CX_ON, the triads for the node and all its children will be displayed in the CIMServer graphics window.

RETURN VALUES

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

SEE ALSO

CxFrameSwitch, CxGetFrameSwitch
CxStructReflectivity

Sets reflectivity for a branch of nodes

SYNOPSIS
#include <code/robpac.h>
long CxReflectivity(CxServer Server, CxNodeId node, double r, double g, double b)

ARGUMENTS
Server      Server ID
node        Node Id
r           Red component (range: 0.0 < r < 1.0)
g           Green component (range: 0.0 < g < 1.0)
b           Blue component (range: 0.0 < b < 1.0)

DESCRIPTION
This function changes the specular reflectivity coefficients of the given node and the branch made up of its child nodes. The reflectivity of all nodes within the branch with the GEOMETRY attribute will be set. If the component value specified does not fall within the range, it will be set to the nearest boundary value (i.e., value < 0 will be set to 0.0, and value > 1 will be set to 1.0).

RETURN VALUES
This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS
If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

SEE ALSO
CxReflectivity, CxGetReflectivity
CxStructTransparency

Sets transparency level for a branch of nodes

SYNOPSIS
#include <code/robpac.h>
long CxStructTransparency(CxServer Server, CxNodeId node, long level)

ARGUMENTS
Server Server ID
node Node ID
level Transparency level (0 ≤ level ≤ 16, where 16 is maximum transparency)

DESCRIPTION
This function changes the transparency level for the given node and the branch made up of its child nodes. Nodes within the branch must have the GEOMETRY attribute. The transparency level of all nodes within the branch with the GEOMETRY attribute will be set. If the level specified does not fall within the range, it will be set to the nearest boundary value (i.e., level < 0 will be set to 0, and level > 16 will be set to 16).

NOTE: The view type of a node in the structure must be set to CLEAR or CLEAR_EDGED for this function to work. See CxViewType for more information.

RETURN VALUES
This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS
If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE.Receive_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

SEE ALSO
CxTransparency, CxGetTransparency, CxViewType
**CxStructViewSwitch**

Sets the view switch for a branch of elements

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxStructViewSwitch(CxServer Server, CxNodeId ele, long onoff)
```

**ARGUMENTS**

- `Server`: Server ID
- `ele`: Node ID
- `onoff`: Switch (CX_ON or CX_OFF)

**DESCRIPTION**

This function sets the view (on/off) for the given element and the branch made up of its child elements. The view (on/off) of all nodes within the branch with the GEOMETRY attribute will be set. If set to CX_ON, the node and all its children will be displayed in the CIMServer graphics window; otherwise, if set to CX_OFF, the node and all its children will not be displayed.

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the `CxGetErrorNumber` function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

- `CxViewSwitch`, `CxGetViewSwitch`
**CxStructViewType**

Sets the view type for a branch of elements

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxStructViewType(CxServer Server, CxNodeId ele, long type)
```

**ARGUMENTS**

- **Server**  
  Server ID
- **ele**  
  Node ID
- **type**  
  View type (see list in **DESCRIPTION**)

**DESCRIPTION**

This function sets the view type for a given element and the branch made up of its child elements. Listed here are some of the possible view types and their descriptions:

- **CX_CLEAR**  
  Transparent
- **CX_CLEAR_EDGED**  
  Transparent edges
- **CX_FILLED**  
  Filled
- **CX_FILLED_EDGED**  
  Filled edged
- **CX_NORMAL**  
  View type set by `CxSetDefaultView`
- **CX_SHADED**  
  Flat shaded
- **CX_SHADED_EDGED**  
  Flat shaded, edged
- **CX_SMOOTH**  
  Smooth shaded
- **CX_SMOOTH_EDGED**  
  Smooth shaded, edged
- **CX_WIRE**  
  Wire frame

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the `CxGetErrorNumber` function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>
SEE ALSO

CxViewType, CxGetViewType
**CxTransparency**
Sets transparency level for an element

**SYNOPSIS**
#include <code/robpac.h>
long CxTransparency(CxServer Server, CxNodeId ele, long level)

**ARGUMENTS**
- **Server** Server ID
- **ele** Node ID
- **level** Transparency level (range: 0 < level < 16 where 16 is transparent)

**DESCRIPTION**
This function sets the transparency level for the given element. The node must have the GEOMETRY attribute. If the level specified does not fall within the range, it will be set to the nearest boundary value (i.e., level < 0 will be set to 0, and level > 16 will be set to 16).

**NOTE:** The view type must be set to CXCLEAR or CX_CLEAR_EDGED for this function to work.

**RETURN VALUES**
This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**
If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>Node does not have the GEOMETRY attribute.</td>
</tr>
</tbody>
</table>

**SEE ALSO**
CxStructTransparency, CxGetTransparency
CxViewSwitch
Sets the view switch for an element

SYNOPSIS
#include <code/robpac.h>
long CxViewSwitch (CxServer Server, CxNodeId ele, long onoff)

ARGUMENTS
Server Server ID
ele Node ID
onoff Switch (CX_ON or CX_OFF)

DESCRIPTION
This function sets the view (on/off) for the given element. The node must have the GEOMETRY attribute. If set to CX_ON, the node will be displayed in the CIMServer graphics window; otherwise, if set to CX_OFF, the node will not be displayed.

RETURN VALUES
This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS
If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>Node does not have the GEOMETRY attribute.</td>
</tr>
</tbody>
</table>

SEE ALSO
CxStructViewSwitch, CxGetViewSwitch
**CxViewType**

Sets the view type for an element

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxViewType(CxServer Server, CxNodeId ele, long type)
```

**ARGUMENTS**

- **Server**  
  Server ID
- **ele**  
  Node ID
- **type**  
  View type (see list in **DESCRIPTION**)

**DESCRIPTION**

This function sets the view type for a given element. The node must have the `GEOMETRY` attribute. Listed here are some of the possible view types and their descriptions:

<table>
<thead>
<tr>
<th>CX_CLEAR</th>
<th>Transparent</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_CLEAR_EDGED</td>
<td>Transparent edges</td>
</tr>
<tr>
<td>CX_FILLED</td>
<td>Filled</td>
</tr>
<tr>
<td>CX_FILLED_EDGED</td>
<td>Filled edged</td>
</tr>
<tr>
<td>CX_NORMAL</td>
<td>View type set by CxSetDefaultView</td>
</tr>
<tr>
<td>CX_SHADED</td>
<td>Flat shaded</td>
</tr>
<tr>
<td>CX_SHADED_EDGED</td>
<td>Flat shaded, edged</td>
</tr>
<tr>
<td>CX_SMOOTH</td>
<td>Smooth shaded</td>
</tr>
<tr>
<td>CX_SMOOTH_EDGED</td>
<td>Smooth shaded, edged</td>
</tr>
<tr>
<td>CX_WIRE</td>
<td>Wire frame</td>
</tr>
</tbody>
</table>

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
<tr>
<td>Error Codes</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>Node does not have the GEOMETRY attribute.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

CxStructViewType, CxGetViewType
Tool Path Display
CxAddTrail

Adds the TRAIL attribute to a node

SYNOPSIS
#include <code/robpac.h>
long CxAddTrail(CxServer Server, CxNodeId node)

ARGUMENTS
Server Server ID

node Node ID that gets the trail attribute

DESCRIPTION
This function adds the TRAIL attribute to a given tree node. A node with the TRAIL attribute will draw a line wherever the node goes which enables the user to see the path that is followed by that node during a move or sequence of moves.

RETURN VALUES
This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS
If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Given node does not exist.</td>
</tr>
<tr>
<td>MACHINE_OUT_OF_MEMORY</td>
<td>Machine out of memory.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

EXAMPLE
The following example adds the TRAIL attribute to node grip_tcf and then sends some moves:

CxServer Server;
CxMechanism mech;
CxNodeId grip_tcf, peg_1_loc;

CxGetNamedNodeId( Server, "grip_tcf", &grip_tcf );
CxGetNamedNodeId( Server, "peg_1_loc", &peg_1_loc );
CxAddTrail( Server, grip_tcf );
CxMoveSingleAxis( mech, 0, 45.0 );
CxMoveToNode( mech, peg_l_loc, grip_tcf );

SEE ALSO

CxDeleteTrail
CxCalcFrameSwitch

Turns the calculation of frames along a trail on or off

SYNOPSIS

#include <code/robpac.h>
long CxCalcFrameSwitch(CxServer Server, CxNodeId node, long onoff, long interval)

ARGUMENTS

Server : Server ID
node : Node ID with TRAIL attribute
onoff : Calculate frames or not. If set to CX_TRUE, the frame positions will be calculated; otherwise, they will not be calculated. Default is CX_FALSE.
interval : The interval at which frames will be calculated if onoff is CX_TRUE. The interval must be greater than or equal to 1.

DESCRIPTION

This function turns on or off the calculation of frames along a trail and sets the interval at which the frames will be placed along the trail if this switch is true. This interval refers to the time step set by CxSetSimrate. For example, if interval=3, then a frame is calculated every 3rd time step. The default is that the frames will not be calculated.

RETURN VALUES

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERROR

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>Node does not have the TRAIL attribute.</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>Specified interval is less than 1.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following code segment shows how this function can be used:

CxServer Server;
CxMechanism mech;
CxNodeId grip_tcf;
.
.
CxAddTrail( Server, grip_tcf );
CxCalcFrameSwitch( Server, grip_tcf, CX_TRUE, 10 );
CxMoveSingleAxis( mech, 0, 90 );
.
.
SEE ALSO

CxGetCalcFrameSwitch, CxSetSimrate
CxCalcTrailSwitch

Toggles the switch for calculating a trail

SYNOPSIS

```c
#include <code/robpac.h>
long CxCalcTrailSwitch(CxServer Server, CxNodeId node, long onoff)
```

ARGUMENTS

- `Server`: Server ID
- `node`: Node ID with TRAIL attribute
- `onoff`: If `CX_TRUE`, the trail will be calculated. Default is `CX_TRUE`.

DESCRIPTION

This function turns on or off the calculation of a trail. If `CX_TRUE`, the trail will be calculated; otherwise, it will not. The default is `CX_TRUE`.

RETURN VALUES

This function returns 0 if successful; otherwise, -1 (`CX_ERROR`) is returned.

ERRORS

If the function returns an error condition, the error code can be obtained by using the `CxGetErrorMessage` function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CX_MESSAGE_SEND_FAILED</code></td>
<td>Error in sending message.</td>
</tr>
<tr>
<td><code>CX_MESSAGE_RECEIVE_FAILED</code></td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td><code>CX_NODE_NOT_FOUND</code></td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td><code>CX_ATTRIBUTE_NOT_FOUND</code></td>
<td>Node does not have the TRAIL attribute.</td>
</tr>
<tr>
<td><code>CX_NODE_IS_CUT_OUT</code></td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td><code>CX_INVALID_NODE_ID</code></td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

SEE ALSO

`CxGetCalcTrailSwitch`
CxClearTrail

Clears a trail and causes a new trail to begin from the current position.

SYNOPSIS

```c
#include <code/robpac.h>
long CxClearTrail(CxServer Server, CxNodeId node)
```

ARGUMENTS

- **Server**: Server ID
- **node**: Node ID with TRAIL attribute

DESCRIPTION

This function erases a trail that has previously been drawn and starts the trail over again so that the next robot move will become the first move of the new trail.

RETURN VALUES

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS

If the function returns an error condition, the error code can be obtained by using the CxGetErrorCode function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>Node does not have the TRAIL attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following code segment shows how this function can be used:

```c
main ()
{
    CxServer Server;
    CxMechanism mech;
    CxNodeId, grip_tcf, peg_1_loc, peg_2_loc;
    ...
    CxAddTrail( Server, grip_tcf );
    CxMoveSingleAxis( mech, 0, 90.0 );
    CxMoveToNode( mech, peg_1_loc, grip_tcf );
}```
CxClearTrail( Server, grip_tcf );
CxMoveToNode( Server, peg_2_loc, grip_tcf);
.
.
.
}

SEE ALSO

CxSetNumTrailMoves
CxDeleteTrail

Deletes the TRAIL attribute from a node

SYNOPSIS

```c
#include <code/robpac.h>
long CxDeleteTrail(CxServer Server, CxNodeId node)
```

ARGUMENTS

- `Server`  
  Server ID
- `node`  
  Node from which to delete the TRAIL attribute

DESCRIPTION

This function deletes the TRAIL property from a given tree node.

RETURN VALUES

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS

If the function returns an error condition, the error code can be obtained by using the CxGetErrrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Tree node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>Node does not have the TRAIL attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

EXAMPLE

```c
main ()
{
    CxServer server ;
    CxMechanism Mech ;
    CxNodeId rob_id, tcf_id ;
    .
    server = CxOpenServer ( myServer, CX_SMEM, 0 ) ;
    CxGetNamedNodeId ( server, "s100", &rob_id ) ;
    CxGetNamedNodeId ( server, "grip_tcf", tcf_id ) ;
    .
    Mech = CxOpenMechanism ( myServer, rob_id, CX_CONTROL ) ;
    CxAddTrail ( server, tcf_id ) ;
```
CxDeleteTrail (server, tcf_id);

SEE ALSO

CxAddTrail
CxGetCalcFrameSwitch
Determines if frames are being calculated

SYNOPSIS
#include <code/robpac.h>
long CxGetCalcFrameSwitch(CxServer Server, CxNodeId node, long *onoff, long *interval)

ARGUMENTS
Server Server ID
node Node ID with TRAIL attribute
onoff Frames calculated or not
interval Interval between the frames along a trail

DESCRIPTION
This function returns the frame calculation information. If onoff is CX_TRUE, the frames are being calculated. interval contains the interval between the frames along the trail. This interval refers to the time step set by CxSetSimrate. For example, if the interval=3, then a frame is calculated every 3rd time step.

RETURN VALUES
This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS
If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>Node does not have the TRAIL attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

EXAMPLE

main ()
{
    CxServer Server;
    CxNodeId tcf;
    long interval, onoff;
    .

CxGetCalcFrameSwitch( Server, tcf, &onoff, &interval );
if( !onoff || interval < 10 )
    CxCalcFrameSwitch( Server, tcf, CX_TRUE, 10 );

SEE ALSO
    CxCalcFrameSwitch
CxGetCalcTrailSwitch

Determines if the trail is being calculated

SYNOPSIS

#include <code/robpac.h>
long CxGetCalcTrailSwitch(CxServer Server, CxNodeId node, long *onoff)

ARGUMENTS

Server       Server ID
node         Node ID with the TRAIL attribute
onoff        Trail being calculated or not

DESCRIPTION

This function returns trail calculation information. If onoff is CX_TRUE, the trail is being calculated; otherwise, it is not.

RETURN VALUES

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>Node does not have the TRAIL attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

EXAMPLE

main ()
{
  CxServer myServer ;
  CxMechanism Mech ;
  CxNodeId s100_id, grip_tcf ;

  myServer = CxOpenServer ( "my_server_name", CX_SMEM, 0 ) ;
  CxGetNamedNodeId ( myServer, "s100", &s100_id ) ;
  CxGetNamedNodeId ( myServer, "grip_tcf", &grip_tcf ) ;
  Mech = CxOpenMechanism ( myServer, s100, CX_CONTROL ) ;
CxAddTrail ( myServer, grip_tcf ) ;
CxSetCalcTrailSwitch ( myServer, grip_tcf, CX_TRUE ) ;

CxGetCalcTrailSwitch ( myServer, grip_tcf, &on_off ) ;
if ( on_off == CX_TRUE )
{
    CxCalcFrameSwitch ( myServer, grip_tcf, CX_TRUE, 50 ) ;
}

SEE ALSO
    CxCalcTrailSwitch
CxGetMoveDraw

Gets the move draw information of a trail node

SYNOPSIS
#include <code/robpac.h>
long CxGetMoveDraw(CxServer Server, CxNodeId node, long *movedraw)

ARGUMENTS
Server  Server ID
node    Node ID with the TRAIL attribute
movedraw If CX_TRUE, the trail will be drawn; otherwise, it will not.

DESCRIPTION
This function tells the user if the trail is being drawn or if it is just in move mode. It is possible that the trail is being calculated, but the user doesn’t want a portion of it drawn. The trail will not be drawn for those moves where movedraw is set to CX_FALSE.

RETURN VALUES
This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS
If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>Node does not have the TRAIL attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

EXAMPLE
The following example shows how this function is to be used:

```c
main ()
{
  CxServer Server;
  CxNodeId tcf;
  long movedraw;
  
  /* Application code */
}
```
CxGetMoveDraw( Server, tcf, &movedraw )
if( movedraw )
   CxSetMoveDraw( Server, tcf, CX_FALSE );
.
.
.
}

SEE ALSO

CxSetMoveDraw
**CxGetNumTrailMoves**

Gets the maximum number of moves that will be kept track of in the trail

**SYNOPSIS**

```
#include <code/robpac.h>
long CxGetNumTrailMoves(CxServer Server, CxNodeId node, long *num)
```

**ARGUMENTS**

- **Server**  
  Server ID
- **node**  
  Node ID with the TRAIL attribute
- **num**  
  Number of moves kept track of in the trail

**DESCRIPTION**

This function gives the user the maximum number of moves that will be kept track of in the trail. Once this number is reached, the beginning of the trail is erased and the space reallocated for the next part of the trail. The higher the number of trail moves, the farther the simulated mechanism can move without starting to erase the beginning of the trail.

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>Node does not have the TRAIL attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

```c
main ()
{
  CxServer Server;
  CxMechanism mech;
  CxNodeId tcf;
  long num;
...
```
CxGetNumTrailMoves( Server, tcf, &num );
if( num < 100 )
    CxMoveSingleAxis( mech, 0, 45.0 );
else
    CxMoveSingleAxis( mech, 0, 150.0 );
.
.
}

SEE ALSO

CxSetNumTrailMoves
**CxGetRefNode**

Gets the node that the trail is drawn in reference to

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxGetRefNode(CxServer Server, CxNodeId node, CxNodeId *ref_node)
```

**ARGUMENTS**

- **Server**  
  Server ID
- **node**  
  Node ID with TRAIL attribute
- **ref_node**  
  Node to which the trail is drawn in reference

**DESCRIPTION**

This function returns the node ID of the node to which the trail is referenced. If this node moves relative to the node containing the trail attribute, the trail is drawn to reflect this change in relative position.

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>Node does not have the TRAIL attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

```c
#include <code/robpac.h>

int main ( int argc, char **argv )
{
    CxServer myServer ;
    CxMechanism Mech ;
    .
    /* Open server and mechanism */
```
myServer = CxOpenServer ( "my_server_name", CX_SMEM, 0); 
.
Mech = CxOpenMechanism ( myServer, s100_id, CX_CONTROL );
CxAddTrail ( myServer, grip_tcf ); 
.
/* Get the trail reference node */
CxGetRefNode ( myServer, grip_tcf, &ref_node );
.
CxRobpacExit ( ) ;
.
}

SEE ALSO

CxSetRefNode
**CxGetTrailColor**

Gets the "rgb" components of the trail color

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxGetTrailColor(CxServer Server, CxNodeId node, double *r, double *g, double *b)
```

**ARGUMENTS**

- **Server** Server ID
- **node** Node ID with the TRAIL attribute
- **r** The trail color's red component
- **g** The trail color's green component
- **b** The trail color's blue component

**DESCRIPTION**

This function gets the red, green and blue components that make up the trail color. These values will be between 0.0 and 1.0 inclusive. The default color is the same color as the nearest ancestor with geometry. If the node does not have an ancestor with geometry, the trail will be white.

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>Node does not have the TRAIL attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

CxSetTrailColor
CxGetTrailViewSwitch
Determines if the trail is turned on or off (can be seen)

SYNOPSIS
#include <code/robpac.h>
long CxGetTrailViewSwitch(CxServer Server, CxNodeId node, long *lonoff,
long *fronoff)

ARGUMENTS
Server    Server ID
node      Node ID with the TRAIL attribute
lonoff    Is the view of the trail on or off
fronoff   Is the view of the frames on or off

DESCRIPTION
This function gets the state of the view for both the trail and the frames. If the view of the trail is on, the variable
lonoff will be CX_TRUE. If the view of the frames is on, the variable fronoff will be CX_TRUE. If either of
the views are off, the respective variable will be returned as CX_FALSE.

RETURN VALUES
This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS
If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber
function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>Node does not have the TRAIL attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

SEE ALSO
CxSetTrailViewSwitch
CxSetMoveDraw
Sets the move draw information of a trail node

SYNOPSIS
#include <code/robpac.h>
long CxSetMoveDraw(CxServer Server, CxNodeId node, long movedraw)

ARGUMENTS
Server       Server ID
node         Node ID with the TRAIL attribute
movedraw     If CX_TRUE, the trail will be drawn; otherwise, it will not.

DESCRIPTION
This function sets whether the trail is being drawn while the robot moves. It is possible that the trail is being calculated, but the user doesn’t want a portion of it drawn. The trail will not be drawn for those moves where movedraw is set to CX_FALSE. The default is CX_TRUE.

RETURN VALUES
This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS
If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>Node does not have the TRAIL attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

EXAMPLE

main()
{
  CxServer Server;
  long movedraw;
  CxNodeId tcf;
  .
  CxGetMoveDraw( Server, tcf, &movedraw )
  if( movedraw )
SEE ALSO

CxGetMoveDraw
**CxSetNumTrailMoves**

Sets the maximum number of moves that will be kept track of in the trail

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxSetNumTrailMoves(CxServer Server, CxNodeId node, long num)
```

**ARGUMENTS**

- **Server**  
  Server ID
- **node**  
  Node ID with the TRAIL attribute
- **num**  
  Number of moves kept track of in the trail

**DESCRIPTION**

This function sets the maximum number of moves that will be kept track of in the trail. Once this number is reached, the beginning of the trail is erased and the space reallocated for the next part of the trail. The higher the number of trail moves, the farther the simulated mechanism can move without starting to erase the beginning of the trail.

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>Node does not have the TRAIL attribute.</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>Number must be greater than 0.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

```c
main ()
{
    CxServer Server;
    CxNodeID tcf;
    long num;
    .
```
CxGetNumTrailMoves( Server, tcf, &num );
if( num < 1000 )
    CxSetNumTrailMoves( Server, tcf, 1000 );
.
.
.
}

SEE ALSO

CxGetNumTrailMoves
### CxSetRefNode

Causes a trail to move relative to a reference node

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxSetRefNode(CxServer Server, CxNodeId node, CxNodeId ref_node)
```

**ARGUMENTS**

- **Server**
  - Server ID
- **node**
  - Node ID with the TRAIL attribute
- **ref_node**
  - Node to which the trail is drawn in reference

**DESCRIPTION**

This function sets the node to which the trail is referenced. If this node moves relative to the node containing the trail attribute, the trail is drawn to reflect this change in relative position.

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

If the function returns an error condition, the error code can be obtained by using the CxGetErrrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>Node does not have the TRAIL attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

CxGetRefNode
CxSetTrailColor

Sets the "rgb" components of the trail color

SYNOPSIS

#include <code/robpac.h>
long CxSetTrailColor(CxServer Server, CxNodeId node, double r, double g, double b)

ARGUMENTS

Server    Server ID
node      Node ID with the TRAIL attribute
r         The trail color's red component (range: 0.0 ≤ r ≤ 1.0)
g         The trail color's green component (range: 0.0 ≤ g ≤ 1.0)
b         The trail color's blue component (range: 0.0 ≤ b ≤ 1.0)

DESCRIPTION

This function sets the red, green and blue components that make up the trail color. These values must be between 0.0 and 1.0, inclusive.

RETURN VALUES

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>Node does not have the TRAIL attribute.</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>One of the components is outside the range between 0.0 and 1.0, inclusive.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

SEE ALSO

CxGetTrailColor
**CxSetTrailViewSwitch**

Turns on or off the switch that determines whether the trail can be seen or not

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxSetTrailViewSwitch(CxServer Server, CxNodeId node, long lonoff, long fronoff)
```

**ARGUMENTS**

- **Server**: Server ID
- **node**: Node ID with the TRAIL attribute
- **lonoff**: Is the view of the trail on or off
- **fronoff**: Is the view of the frame on or off

**DESCRIPTION**

This function sets the state of the view for both the trail and the frames. If `lonoff` is CX_TRUE, the trail is turned on (it can be seen). If `fronoff` is CX_TRUE the frames will be turned on (they can be seen). If either of the variables is set to CX_FALSE, the respective view will be turned off.

**NOTE**: If the `CxCalcTrailSwitch` or `CxCalcFrameSwitch` is turned off, the user will not be able to see the respective view no matter what the variable is set to.

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the `CxGetErrorNumber` function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>Specified node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>Node does not have the TRAIL attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Given node has been cut out.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

CxGetTrailViewSwitch
Matrix
A matrix can be expressed either in column form or in row form. A homogeneous matrix in column form may be written as follows:

\[
\begin{bmatrix}
    nx & ox & ax & px \\
    ny & oy & ay & py \\
    nz & oz & az & pz \\
    0 & 0 & 0 & 1
\end{bmatrix}
\]

The same matrix in row form may be written as follows:

\[
\begin{bmatrix}
    nx & ny & nz & 0 \\
    ox & oy & oz & 0 \\
    ax & ay & az & 0 \\
    px & py & pz & 1
\end{bmatrix}
\]

Notice that a matrix in column form is a transpose of its corresponding matrix in row form.

Traditionally, matrices are expressed in column form. Most math and robotics textbooks follow this convention. The C language stores multi-dimensional arrays in row form, so it seems natural to use row form when programming in C.

Unless otherwise specified, all matrices in this library are passed in and returned in row form.

For example, \( T_c \) above would be represented as:

\[
\begin{aligned}
    \{ \{ nx, ox, ax, px \}, \\
    \{ ny, oy, ay, py \}, \\
    \{ nz, oz, az, pz \}, \\
    \{ 0, 0, 0, 1 \} \}
\end{aligned}
\]

Both forms are valid as long as the underlying principles of matrix algebra are not violated.

**WARNING!**: The functions in this library require that all angles be in radians. \texttt{CxSetUnit} has no effect for the functions listed in this library.
**CxCompareMat**

Compares the positional and/or rotational portions of two pose matrices

**SYNOPSIS**

```c
#include <code/robconst.h>
#include <code/matrix.h>
long CxCompareMat (CxMatrix mat1, CxMatrix mat2, CxMatrix rel_mat, double position_tol, double screw_tol, long comp_type)
```

**ARGUMENTS**

- **mat1**: First of two matrices to be compared
- **mat2**: Second of two matrices to be compared
- **rel_mat**: Returns the relative pose matrix from mat1 to mat2
- **position_tol**: Distance between two pose matrices allowed before they are no longer considered identical
- **screw_tol**: Amount of rotation in radians between the two pose matrices allowed before they are no longer considered identical
- **comp_type**: Specifies one of three comparison options, which are:
  - **CX_BOTH**: Both positional and rotational differences
  - **CX_POS_ONLY**: Check positional difference only
  - **CX SCREW ONLY**: Check rotational difference only

For rel_mat to have meaning, both mat1 and mat2 must be relative to the same frame. In addition, they must both be homogenous matrices.

**DESCRIPTION**

This function checks two matrices to see if they are the same to within specified tolerances. The user can check the entire matrix, or just the positional or rotational information.

**RETURN VALUE**

This function returns **CX_TRUE** (1) if the two matrices are considered identical by the specified comp_type and tolerances; otherwise, the function returns **CX_FALSE** (0).

**EXAMPLE**

The following sample code segment determines if the two matrices are within specified tolerances (0.1 unit in position and 0.01 radians).

```c
CxMatrix mat1, mat2, rel_mat;
/* get or assign values to mat1 and mat2 */
```
if( CxCompareMat(mat1, mat2, rel_mat, 0.1, 0.01, CX_BOTH) )
    fprintf(stderr, "mat1 and mat2 are two identical matrices.\n");
else
    fprintf(stderr, "mat1 and mat2 are two different matrices.\n");

SEE ALSO

CxRlmat
**CxCross**

Computes the cross product of two vectors

**SYNOPSIS**

```c
#include <code/matrix.h>
void CxCross (CxVector vec_x, CxVector vec_y, CxVector vec_z)
```

**ARGUMENTS**

- **vec_x**  
  First vector in the cross product
- **vec_y**  
  Second vector in the cross product
- **vec_z**  
  Returns the resulting cross product vector

**DESCRIPTION**

This function takes two vectors as input and returns as output the cross product (vector product) of the two input vectors.

The vector cross product is order-sensitive. For example, if `vec_x` were switched with `vec_y`, the magnitude of the resulting vector would be the same, but its direction would be reversed. The resulting vector is normal to the other two (source) vectors.

The cross product of `vec_x` with `vec_y` is calculated by taking the following determinant:

```
\[
\begin{vmatrix}
  \hat{i} & \hat{j} & \hat{k} \\
\end{vmatrix}
\]
```

**EXAMPLE**

The following code fragment defines two vectors, `vec_x` and `vec_y`, and then determines a vector `vec_z` as the cross product of `vec_x` with `vec_y`.

```c
CxVector vec_x, vec_y, vec_z;
vec_x[0] = 1.3;
vec_x[1] = 2.6;
vec_x[2] = -4.3;
vec_y[0] = 8.4;
vec_y[1] = -0.4;
vec_y[2] = 7.7;
CxCross (vec_x, vec_y, vec_z);
```

The following are the values returned by `vec_z`:

- `vec_z[0] = 18.3`
- `vec_z[1] = -46.13`
CxDistan

Gets the distance between two points

SYNOPSIS

```c
#include <code/matrix.h>
double CxDistan(CxVector vec1, CxVector vec2)
```

ARGUMENTS

- vec1: Coordinates for the first point
- vec2: Coordinates for the second point

DESCRIPTION

This function takes two vectors, representing the positions of two points, as input and returns the distance between the two specified points. The following equation illustrates how the distance is calculated where `vec1` and `vec2` are two vectors containing the coordinate information:

\[
\text{distance} = \sqrt{(vec1[0] - vec2[0])^2 + (vec1[1] - vec2[1])^2 + (vec1[2] - vec2[2])^2}
\]

RETURN VALUE

This function returns the distance between the two specified points.

EXAMPLE

The first example uses the `CxFindtr` function (from the “Matrix” library) to extract the position vector for each of two pose matrices. The `CxDistan` function is then called to determine the distance between the two frames (located at the tip of each vector).

```c
CxMatrix pose_mat1, pose_mat2;
CxVector trans_vec1, trans_vec2;
double distance;

/* extract position vector from pose matrix */
CxFindtr (pose_mat1, trans_vec1);
CxFindtr (pose_mat2, trans_vec2);

/* find distance between two frames */
distance = CxDistan(trans_vec1, trans_vec2);
```

The second example simply defines two vectors, `vec1` and `vec2`, and then calculates the distance between the two frames.

```c
CxVector vec1, vec2;
double distance;

/* assign values to vec1 and vec2 */
```
vec1[0] = 2.1;
vec1[1] = 4.7;
vec1[2] = -9.5;
vec2[0] = -3.5;
vec2[1] = 0.95;
vec2[2] = 1.6;

/* determine the distance between two points */
distance = distan (vec1, vec2);

NOTE: The distance found in the second example is 12.99.
Returns the scaled dot product of two vectors

SYNOPSIS

#include <code/matrix.h>
double CxDotp (double factor, CxVector vec1, CxVector vec2)

ARGUMENTS

factor  Factor by which the scalar result of the dot product is divided
vec1    One of the two vectors used in the dot product
vec2    One of the two vectors used in the dot product

DESCRIPTION

This function takes two vectors as input and returns the dot product (scalar product) of the two divided by a specified factor as output.

The following equation illustrates the calculation of the scaled dot product.

\[ \text{CxDotp} = \frac{\text{vec1} \cdot \text{vec2}}{\text{factor}} \]

where \( \text{vec1} \cdot \text{vec2} \) is calculated using the following equation:

\[ \text{vec1} \cdot \text{vec2} = \text{vec1}[0] \times \text{vec2}[0] + \text{vec1}[1] \times \text{vec2}[1] + \text{vec1}[2] \times \text{vec2}[2] \]

If the absolute value of factor is less than CX_EPSILONSS (defined in <code/const.h.>) then factor is set to 1.0 before dividing.

RETURN VALUE

This function returns the scalar value of the dot product divided by the specified factor.

EXAMPLE

The following code fragment defines two vectors, vec1 and vec2, takes their dot product, and divides the product by two.

double val;
CxVector vec1, vec2;

vec1[0] = 2.0;
vec1[1] = 3.5;
vec1[2] = 7.4;

vec2[0] = -3.0;
vec2[1] = 5.5;
vec2[2] = -1.5;

val = CxDotp(2.0, vec1, vec2);

In this case, the final value is 1.075 (dot product 2.15 divided by factor of 2).

SEE ALSO

CxDots
**CxDots**

Returns the dot product of two vectors

**SYNOPSIS**

```c
#include <code/matrix.h>
double CxDots(CxVector vec1, CxVector vec2)
```

**ARGUMENTS**

- `vec1` One of the two vectors used in the dot product
- `vec2` One of the two vectors used in the dot product

**DESCRIPTION**

This function takes two vectors as input and returns the dot product (scalar product) of the two vectors.

**EXAMPLE**

The following code fragment simply defines two vectors, `vec1` and `vec2`, and takes their dot product.

```c
CxVector vec1, vec2;
double val;

vec1[0] = 2.0;
vec1[1] = 3.5;
vec1[2] = 7.4;

vec2[0] = -3.0;
vec2[1] = 5.5;
vec2[2] = -1.5;

val = CxDots(vec1, vec2);
```

In this case, the final value is 2.15.

**SEE ALSO**

`CxDotp`
CxEuler

Finds ordered Euler angles (Z Y Z) from a pose matrix

SYNOPSIS

```c
#include <code/matrix.h>
void CxEuler (CxMatrix mat, CxVector angles)
```

ARGUMENTS

- `mat`: Pose matrix to be analyzed
- `angles`: Returns the ordered Euler angles in radians (Z Y Z)

DESCRIPTION

This function takes a pose matrix and returns the rotational information in the form of three angles, representing rotations about (in order) the frame's own Z, Y, and Z (again) axes.

This set of three angles, Euler angles, is a convenient and compact way to represent the same information contained in the rotational portion of a pose matrix. Some robots require the rotational information to be in this form.

The Euler angle convention describes a rotation first about the Z axis followed by a rotation about the just rotated Y axis and then a second and final rotation about the twice-rotated Z axis.

EXAMPLE

The following sample code extracts Euler angles from a given 4x4 matrix and prints the Euler angles in degrees.

```c
#define RAD_TO_DEG 57.29578
CxMatrix mat={ {0.707, 0.0, 0.707, 1.0},
               {-0.3535, 0.866, 0.3535, 3.0},
               {-0.6124, -0.50, 0.6124, 15.0},
               {0.0, 0.0, 0.0, 1.0}};
CxVector angles;

/* extract Euler angles from mat */
CxEuler (mat, angles);

/* print angles out to screen */
fprintf (stderr, "The corresponding Euler angles for the \n"  "given matrix mat are: %f %f %f degrees\n",
        angles[0] * RAD_TO_DEG, angles[1] * RAD_TO_DEG, angles[2] * RAD_TO_DEG);
```

SEE ALSO

CxRolyaw, CxXyzang
# CxFindtr

Retrieves the translation vector from a pose matrix

## SYNOPSIS

```c
#include <code/matrix.h>
void CxFindtr(CxMatrix mat, CxVector trans_vec)
```

## ARGUMENTS

- **mat**
  - Input pose matrix
- **trans_vec**
  - Output translation vector from mat

## DESCRIPTION

This function takes a pose matrix as input and returns the translation (position or displacement) information as a vector. The `CxFindtr` function allows the user to access the translation (or displacement) vector of a pose matrix directly, without having to access the entire matrix.

## EXAMPLE

This example gets the translation vector from the pose matrix `j2mat`, where `j2mat` is given by the following matrix (in row form):

\[
\begin{bmatrix}
1.0 & 0.0 & 0.0 & 360.0 \\
0.0 & 0.71 & -0.71 & 400.0 \\
0.0 & 0.71 & 0.71 & 300.0 \\
360. & 400. & 300. & 1.00 \\
\end{bmatrix}
\]

```c
CxMatrix j2mat={{1.0, 0.0, 0.0, 360.0},
{0.0, 0.71, -0.71, 400.0},
{0.0, 0.71, 0.71, 300.0},
{360.0, 400.0, 300.0, 1.00}};
CxVector j2_trans_vec;
/* extract translation vector */
CxFindtr (j2mat, j2_trans_vec);
```

The values returned in `j2_trans_vec` are the following:

- `j2_trans_vec[0] = 360.`
**CxHForm**

Transforms a homogeneous vector by a matrix

**SYNOPSIS**

```c
#include <code/matrix.h>
void CxHForm(CxMatrix mat, CxVector in_vec, CxVector out_vec)
```

**ARGUMENTS**

- **mat**
  - Matrix to be used in the vector transformation
- **in_vec**
  - Vector to be transformed
- **out_vec**
  - Returns the transformed vector

**DESCRIPTION**

This function takes as input a vector and a 4x4 matrix (such as a pose matrix in row form), and returns a vector that is the input vector transformed using the input matrix. The following equations are implemented in the function CxHForm:

\[
\begin{bmatrix}
\text{tmp[0]} \\
\text{tmp[1]} \\
\text{tmp[2]} \\
\text{tmp[3]}
\end{bmatrix}^t = \begin{bmatrix}
\text{in_vec[0]} \\
\text{in_vec[1]} \\
\text{in_vec[2]} \\
1.0
\end{bmatrix}^t \begin{bmatrix}
r11 & r12 & r13 & r14 \\
r21 & r22 & r23 & r24 \\
r31 & r32 & r33 & r34 \\
r41 & r42 & r43 & r44
\end{bmatrix}
\]

\[
\begin{bmatrix}
\text{out_vec[0]} \\
\text{out_vec[1]} \\
\text{out_vec[2]}
\end{bmatrix} = \begin{bmatrix}
\text{tmp[0]} / \text{tmp[3]} \\
\text{tmp[1]} / \text{tmp[3]} \\
\text{tmp[2]} / \text{tmp[3]}
\end{bmatrix}
\]
**CxIdentm**

Initializes a 4x4 matrix to an identity matrix

**SYNOPSIS**

```c
#include <code/matrix.h>
void CxIdentm (CxMatrix mat)
```

**ARGUMENTS**

- `mat`  Matrix to be initialized to an identity matrix

**DESCRIPTION**

This function takes a 4x4 matrix as its input and initializes it as an identity matrix. This function only initializes 4x4 matrices. The matrix, `mat`, is thus stored as the following:

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

mat =

1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
**CxInvNxn**

Calculates the inverse of a general nxn matrix

**SYNOPSIS**

```c
#include <code/const.h>
#include <code/matrix.h>
long CxInvNxn(double *mat, double *mat_inv, long n)
```

**ARGUMENTS**

- **mat** Matrix to be inverted
- **mat_inv** Returns the inverse of the specified matrix
- **n** Dimension of the matrix to be inverted, i.e. the number of columns it has; n must be $\geq 1$, otherwise, CX_ERROR will be returned.

**DESCRIPTION**

This function calculates the inverse of any square $(n \times n)$ matrix.

**NOTE:** The function CxInvmat is designed to invert pose matrices, and is more efficient at that task than CxInvNxn. Use CxInvmat for pose matrices and CxInvNxn for other matrices.

**RETURN VALUE**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**EXAMPLE**

The following example uses CxInvNxn to get an inverse matrix of a $6 \times 6$ matrix. It prints out an error message if the matrix is singular; otherwise, it prints the inverse matrix to the screen.

```c
int i;
double *mat, *new_mat;

/* allocate memory for 6x6 matrices */
mat = (double *) calloc (36, sizeof( double));
new_mat = (double *) calloc (36, *sizeof( double));
/* assign values to mat */
.
if (CxInvNxn (mat,new_mat,6) == CX_ERROR) {
    fprintf (stderr,"The given matrix is singular.\n");
} else {
    printf ("The given matrix is non-singular.\n");
    printf ("The inverse matrix is:\n");
    CxPrintMxn (new_mat, 6, 6);
}
```

**SEE ALSO**

CxInvmat
CxInvmat

Calculates the inverse of a pose matrix

SYNOPSIS

```c
#include <code/matrix.h>
void CxInvmat(CxMatrix mat, CxMatrix mat_inv)
```

ARGUMENTS

<table>
<thead>
<tr>
<th>mat</th>
<th>Pose matrix to be inverted</th>
</tr>
</thead>
<tbody>
<tr>
<td>mat_inv</td>
<td>Returns the inverse of the specified pose matrix</td>
</tr>
</tbody>
</table>

DESCRIPTION

This function calculates the inverse of a pose matrix. The function takes advantage of several unique properties of pose matrices and is therefore more efficient at calculating the inverse. The general inverse matrix function, CxInvNxn, should be used for calculating the inverse of matrices that are not pose matrices.

EXAMPLE

The following code segment determines the transformation matrix of frame a relative to frame b. Both frames a and b are defined with respect to the world coordinate frame.

```c
CxMatrix a, b, b_inv, a_to_b;
/* get or load values for matrices a and b */
. .
/* find the inverse matrix of b */
CxInvmat (b, b_inv);
/* determines frame a with respect to frame b */
CxMul4x4 (b_inv, a, a_to_b);

a_to_b can also be found using the CxRlmat function as shown here.

CxRlmat (a, b, a_to_b);
```

SEE ALSO

CxInvNxn
CxLuDecomp
Decomposes a matrix into upper and lower submatrices

SYNOPSIS
#include <code/matrix.h>
long CxLuDecomp (double **a, long n, long *indx, long *d)

ARGUMENTS
a        Square matrix to be LU decomposed, also returns results here
n        Dimension of matrix a; must be > 0
indx     Records the row permutation affected by the partial pivoting
d        +1 or -1 (value depends on even or odd number of row interchanges)

DESCRIPTION
This function assumes some knowledge and experience with lower and upper triangular (LU) matrix algebra on
the part of the user.
This function decomposes the array $a$ (matrix $a$, stored in row form) into upper and lower triangular submatrices
$\alpha$ and $\beta$, where $\alpha$ is the region above the diagonal ($\alpha[i][i] = 1.0$) and $\beta$ is the region below the diagonal including
the diagonal. For more information, see Numerical Recipes, by Press, Flannery, Teukolsky, and Vetterling.

$$
\begin{bmatrix}
\beta_{11} & 0 & 0 & 0 \\
\beta_{21} & \beta_{22} & 0 & 0 \\
\beta_{31} & \beta_{32} & \beta_{33} & 0 \\
\beta_{41} & \beta_{42} & \beta_{43} & \beta_{44}
\end{bmatrix}
\begin{bmatrix}
1 & \alpha_{12} & \alpha_{13} & \alpha_{14} \\
0 & 1 & \alpha_{23} & \alpha_{24} \\
0 & 0 & 1 & \alpha_{34} \\
0 & 0 & 0 & 1
\end{bmatrix}
= 
\begin{bmatrix}
a_{11} & a_{12} & a_{13} & a_{14} \\
a_{21} & a_{22} & a_{23} & a_{24} \\
a_{31} & a_{32} & a_{33} & a_{34} \\
a_{41} & a_{42} & a_{43} & a_{44}
\end{bmatrix}
$$

The array $a$ is overwritten and replaced with $\alpha$ and $\beta$. The parameter $\text{indx}$ is created to keep track of the row
permutations, and $d$ keeps track of the parity ($-1$ if an odd number of row switches, $1$ if an even number of
switches), while $n$ is the number of rows and columns in matrix $a$.

The matrix $a$ returned is filled with $\alpha$’s and $\beta$’s by rows from top to bottom, and within each row from left to right.

\[
\begin{bmatrix}
\beta_{11} & \alpha_{12} & \alpha_{13} & \alpha_{14} \\
\beta_{21} & \beta_{22} & \alpha_{23} & \alpha_{24} \\
\beta_{31} & \beta_{32} & \beta_{33} & \alpha_{34} \\
\beta_{41} & \beta_{42} & \beta_{43} & \beta_{44}
\end{bmatrix}
\]

RETURN VALUES
This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.
SEE ALSO

CxLuLinSolver, CxLuInvmat, CxLudet
CxLuDet

Finds the determinant of a matrix, using LU decomposition

SYNOPSIS

#include <code/matrix.h>
long CxLuDet (double **mat, long n, double *det)

ARGUMENTS

mat  Square input matrix A
n    Dimension of the matrix A; i.e. the number of columns it has.
det  Determinant of matrix A

DESCRIPTION

This function is used to find the determinant of a matrix, using LU decomposition procedures.

RETURN VALUES

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

EXAMPLE

The following code figures out the determinant of a 3 by 3 matrix:

\[
\begin{bmatrix}
1 & 2 & 4 \\
3 & 5 & 2 \\
0 & 0 & 1
\end{bmatrix}
\]

double **mat, det;
int i;

/* allocate memory for mat */
mat = (double **) calloc (3, sizeof(double *));
for(i=0; i<3; i++) {
    mat[i] = (double *) calloc (3, sizeof(double));
}
/* assign values to matrix mat */
mat[0][0] = 1.0;
mat[0][1] = 2.0;
mat[0][2] = 4.0;
mat[1][0] = 3.0;
mat[1][1] = 5.0;
mat[1][2] = 2.0;
mat[2][0] = mat[2][1] = 0.0;
mat[2][2] = 1.0;

/* find determinant of matrix mat */
CxLuDet (mat, 3, &det);

/* prints out determinant */
printf ("the determinant is %f\n", det);

In this case, the determinant is –1.0.

SEE ALSO
   CxLuDecomp, CxLuLinSolver, CxLuInvMat
CxLuInvmat

Finds the inverse of a square matrix using LU decomposition

SYNOPSIS

#include <code/matrix.h>
long CxLuInvmat (double **mat, long n, double **invmat)

ARGUMENTS

mat  Square input matrix A
n    Dimension of the matrix A; i.e. the number of columns it has
invmat Inverse of matrix A

DESCRIPTION

This function is used to find the inverse of a matrix, using LU decomposition techniques.

RETURN VALUES

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

SEE ALSO

CxLuDecomp, CxLuLinSolver, CxLudet
CxLuLinSolver

Solves linear system $Ax = b$ using LU decomposition

SYNOPSIS

```c
#include <code/matrix.h>
long CxLuLinSolver (double **A, long n, double *b, double *x)
```

ARGUMENTS

- **A**: Input matrix $A$ which must be a square matrix
- **n**: Dimension of the linear system; i.e. the number of columns in $A$
- **b**: Input vector $b$ which must be $n$ elements long
- **x**: Solutions of the linear system; that is, the vector $x$, where $Ax=b$

DESCRIPTION

This function is used to solve linear system $Ax = b$ using LU decomposition procedures.

RETURN VALUES

This function returns $0$ if successful; otherwise, $-1$ (CX_ERROR) is returned.

EXAMPLE

```c
double **mat = {(1.0, 2.0, 4.0), (3.0, 5.0, 2.0), (0.0, 0.0, 1.0)};
double *b = {1.0, 0.0, 3.0};
double x[3];

CxLuLinSolver ( mat, 3, b, x ) ;
printf ( "%lf %lf %lf\n", x[0], x[1], x[2] ) ;
```

The following will be printed:

```
43.000000    -27.000000    3.000000
```

SEE ALSO

CxLuDecomp, CxLuInvmat, CxLudet
CxMakero

Creates a pose matrix with a pure rotation about a principal axis

SYNOPSIS

```
#include <code/matrix.h>
long CxMakero (CxMatrix mat, double angle, char axis)
```

ARGUMENTS

- **mat**: Returns the pose matrix with the specified rotation
- **angle**: Angle in radians of the rotation to be generated
- **axis**: The principle axis to be rotated about must only be one of the following characters: 'X,' 'Y,' 'Z,' 'x,' 'y,' or 'z.'

DESCRIPTION

This function creates a pure rotation (zero translation) 4x4 pose matrix from a specified principal axis of rotation and a specified angle of rotation about that axis. The pose matrix, `mat`, will have one of the following three forms (row form), depending on which axis is specified:

- **X axis** =
  
  \[
  \begin{bmatrix}
  1 & 0 & 0 & 0 \\
  0 & c & s & 0 \\
  0 & -s & c & 0 \\
  0 & 0 & 0 & 1
  \end{bmatrix}
  \]

- **Y axis** =
  
  \[
  \begin{bmatrix}
  c & 0 & -s & 0 \\
  0 & 1 & 0 & 0 \\
  s & 0 & c & 0 \\
  0 & 0 & 0 & 1
  \end{bmatrix}
  \]

- **Z axis** =
  
  \[
  \begin{bmatrix}
  c & s & 0 & 0 \\
  s & c & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & 0 & 1
  \end{bmatrix}
  \]

Here \( s = \sin(\text{angle}) \) and \( c = \cos(\text{angle}) \).
RETURN VALUES

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NON_XYZ_AXIS</td>
<td>Axis specified is not valid.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following code creates a pose matrix box_mat with 90 rotation about the principal X axis. Since all matrix operations require that angles in radians, the angle 90 must be entered in radians.

```c
matrix box_mat;
CxMakero (box_mat, 1.5708, 'X');
```

The box_mat will have following values:

```
box_mat =
\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & -1 & 0 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
```

SEE ALSO

CxMaketr, CxRot, CxXyzmat
**CxMaketr**

Creates a pose matrix with a pure translation

**SYNOPSIS**

```c
#include <code/matrix.h>
void CxMaketr(CxMatrix mat, CxVector vec)
```

**ARGUMENTS**

- **mat** Returns a pose matrix containing the specified translation
- **vec** Translation to incorporate into the pose matrix

**DESCRIPTION**

This function takes a vector (representing a position) as input and returns a $4 \times 4$ pose matrix that describes a pure translation (zero rotation) equivalent to the position specified by the vector.

$$
\text{mat} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
dx & dy & dz & 1
\end{bmatrix}
$$

The pose matrix will have the following form (row form):

where $dx = \text{vec}[0]$, $dy = \text{vec}[1]$, and $dz = \text{vec}[2]$.

**EXAMPLE**

The following program generates a pure translational matrix with the specified position vector. The resultant pure translation matrix will be:

```c
#include <stdio.h>
#include <code/matrix.h>

void main(void)
{
    CxMatrix pure_trans_mat;
    CxVector position;

    position[0] = 0.1;
    position[1] = 0.5;
```
position[2] = 3.2;

CxMakeTr (pure_trans_mat, position);
printf(" Pure translation matrix:\n");
CxPrintM(pure_trans_mat);
}

SEE ALSO

CxMakeR, CxXyzmat
**CxMatequ**

Duplicates a 4x4 matrix

**SYNOPSIS**

```c
#include <code/matrix.h>
void CxMatequ (CxMatrix mat1, CxMatrix mat2)
```

**ARGUMENTS**

- **mat1**  
  Matrix to be duplicated
- **mat2**  
  Returns a duplication of the specified matrix

**DESCRIPTION**

The function `CxMatequ` takes a 4x4 matrix as input and returns another 4x4 matrix that is an exact duplicate of the first. In other words, `mat2 = mat1`.

The function is specifically designed for 4x4 matrices only. The user must write a function to copy a matrix of any other size.

**EXAMPLE**

The following code gets the pose matrix of a box with respect to a table and makes a copy of the matrix for future comparison.

```c
CxMatrix box_pose_mat, copy_box_mat;
char axes[4];
CxVector angle, vec;
CxNodeId box, table;

/* get box’s pose with respect to table*/
CxGetPose (Server, box, table, axes, angle, vec);

/* make box_pose_mat */
CxXyzmat(axes, angle, vec, box_pose_mat);

/* make a copy of box_pose_mat before any changes are made */
CxMatequ (box_pose_mat, copy_box_mat)
```

CxMul4x4

Multiplies two 4x4 matrices

SYNOPSIS

#include <code/matrix.h>

void CxMul4x4 (CxMatrix mat1, CxMatrix mat2, CxMatrix mat_result)

ARGUMENTS

mat1    First matrix in the multiplication operation
mat2    Second matrix in the operation
mat_result Returns the resultant matrix (mat_result = mat1 x mat2)

DESCRIPTION

This function will only multiply two 4x4 matrices. The order of matrix multiplication is important. In other words, mat1 times mat2 does not result in the same matrix as mat2 times mat1.

The function can be used to change the relative frame of a pose matrix. For example, suppose that mat1 is the pose of a table relative to the world reference frame and that mat2 is the pose of a part relative to the table. Then, the product of mat1 times mat2 is a new matrix, which represents the pose of the part relative to the world reference frame.

WARNINGS

For the function CxMul4x4, the matrices mat1 and mat2 must be passed in row form. The resulting matrix mat_result is also created in row form. The function internally switches mat1 and mat2 such that mat2 is pre-multiplied by mat1 in terms of the conventional column form representation.

EXAMPLE

CxMatrix mat1={ {0.707, 0.354, 0.612, 1.0},
                 {0, 0.866, -0.5, 3.0},
                 {-0.707, 0.354, 0.612, 15.0},
                 {0, 0, 0, 1}};

CxMatrix mat2={ {0.123, 0.696, -0.707, 1.0},
                 {-0.985, 0.174, 0.0, 2.0},
                 {0.023, 0.696, 0.707, 3.0},
                 {0, 0, 0, 1}};

CxMatrix mmat ;

CxMul4x4 ( mat1, mat2, mmat ) ;
rm_at will contain the following matrix; shown here in row form

\[
\begin{bmatrix}
0.0 & 0.0 & -1.0 & 0.0 \\
-0.766 & 0.643 & 0.0 & 0.0 \\
0.743 & 0.766 & 0.0 & 0.0 \\
0.013 & 13.658 & 12.898 & 1.0 \\
\end{bmatrix}
\]

SEE ALSO

CxMultMxn
**CxMultMxn**

Multiplies two conformable matrices

**SYNOPSIS**
```
#include <code/matrix.h>
void CxMultMxn (double *mat1, double *mat2, double *mat_result, long m,
                long n, long o)
```

**ARGUMENTS**
- `mat1`: First matrix in the multiplication operation (m x n)
- `mat2`: Second matrix in the operation (n x o)
- `mat_result`: Returns the matrix resulting from the multiplication operation (m x o)
- `m`: Row dimension of `mat1`
- `n`: Column dimension of `mat1` and row dimension of `mat2`
- `o`: Column dimension of `mat2`

**DESCRIPTION**
This function multiplies two general matrices. To multiply two matrices they must be conformable—the column dimension of the first matrix must be the same as the row dimension of the second matrix. The size of the resulting matrix is m by o. The order of matrix multiplication is important; in other words, `mat1` times `mat2` does not result in the same matrix as `mat2` times `mat1`.

**WARNINGS**
Unlike `CxMul4x4`, this function does not care whether its input matrices are in row form or in column form. It only cares if the input matrices are conformable. (i.e. the first matrix must be m by n, and the second matrix must be n by o.) The resulting matrix will then be m by o. The number of columns in the first matrix must equal the number of rows in the second matrix.

**EXAMPLE**
The following code segment post-multiplies `mat1` by `mat2`.

```c
CxMatrix mat1={ {4.0, 2.0, 3.0, 1.0},
               {1.1, 2.0, 5.0, 3.0},
               {4.0, 7.0, 21.0, 15.0},
               {0.0, 0.0, 0.0, 1.0}};
CxMatrix mat2={ {4.0, 2.0, 3.0, 1.0},
               {1.1, 2.0, 5.0, 3.0},
               {4.0, 7.0, 21.0, 15.0},
               {0.0, 0.0, 0.0, 1.0}};
CxMatrix mmat ;
CxMultMxn ( mat1, mat2, mmat, 4, 4, 4 );
```
`mMAt` will contain the following matrix:

\[
\begin{bmatrix}
30.2 & 26.6 & 107.7 & 0 \\
33.0 & 41.2 & 169.0 & 0 \\
85.0 & 118.3 & 488.0 & 0 \\
56.0 & 85.1 & 355.0 & 1
\end{bmatrix}
\]

**SEE ALSO**

`CxMul4x4`
**CxNorm**

Normalizes the rotation portion of a pose matrix

**SYNOPSIS**

```c
#include <code/matrix.h>
long CxNorm (CxMatrix mat)
```

**ARGUMENTS**

- `mat`: Specifies the pose matrix to be normalized and returns the normalized matrix

**DESCRIPTION**

The norm of each of the three vectors in the rotation portion of a pose matrix should equal one. Round-off errors can cause the norm of these vectors to be slightly different from unity after several computational operations; therefore, a pose matrix should be periodically normalized using this function. Note that the matrix is passed and returned through the same parameter.

**RETURN VALUES**

This function returns `-1` (`CX_ERROR`) if the normal of one of the three directional vectors is near zero; otherwise, it returns 0.

**EXAMPLE**

The following code performs matrix multiplication on two given matrices, and normalizes the result matrix.

```c
CxMatrix mat1, mat2, r_mat;
CxMul4x4 (mat1, mat2, r_mat);
CxNorm (r_mat);
```

**SEE ALSO**

`CxNormlz`, `CxNormv`
CxNormlz
Normalizes a vector

SYNOPSIS
#include <code/matrix.h>
long CxNormlz (CxVector vec)

ARGUMENTS
vec Specifies the vector to be normalized and returns the normalized vector

DESCRIPTION
This function normalizes a vector. The magnitude of a normalized vector is one. To normalize a vector, each component of the vector is divided by the original magnitude. The following equation illustrates how a vector is normalized:

$$\text{vec}_\text{norm} = \frac{\text{vec}[0]}{|\text{vec}|} \hat{i} + \frac{\text{vec}[1]}{|\text{vec}|} \hat{j} + \frac{\text{vec}[2]}{|\text{vec}|} \hat{k}$$

where $|\text{vec}|$ is the vector magnitude calculated from the following:

$$|\text{vec}| = \sqrt{(\text{vec}[0])^2 + (\text{vec}[1])^2 + (\text{vec}[2])^2}$$

RETURN VALUES
This function returns -1 (CX_ERROR) if the norm of the given vectors is near zero; otherwise, it returns 0.

EXAMPLE
The following code segment assigns values to a vector, normalizes the vector, and prints the normalized vector to the screen.

CxVector vec;
/* load values to vec */
vec[0] = 2.0;
vec[1] = 3.0;
vec[2] = 6.0

/* normalize vec */
CxNormlz (vec);

/* print out the normalized vector */
printf("The normalized vector is :\n");
vec[0], vec[1], vec[2]);
After normalization, the vector \texttt{vec} has the following values:

\begin{align*}
\text{vec}[0] &= 0.2857; \\
\text{vec}[1] &= 0.4286; \\
\text{vec}[2] &= 0.8571;
\end{align*}

\textbf{SEE ALSO}

\texttt{CxNorm}, \texttt{CxNormv}
CxNormv
Calculates the norm of a vector

SYNOPSIS
#include <code/matrix.h>
double CxNormv (CxVector vec)

ARGUMENTS
vec Input vector

DESCRIPTION
The CxNormv function calculates the norm of a vector.
The norm of a vector (also referred to as the magnitude of a vector) can be defined as:

\[
\text{norm of vector vec} = \sqrt{(\text{vec}[0])^2 + (\text{vec}[1])^2 + (\text{vec}[2])^2}
\]

RETURN VALUES
This function returns the norm of a given vector.

EXAMPLE

    CxVector vec;

    /* load values to vec */
    vec[0] = 2.0;
    vec[1] = 3.0;
    vec[2] = 6.0

    printf("The norm of the given vector is %lf\n", CxNormv (vec));

The norm for vector (2, 3, 6) is 7.

SEE ALSO
CxNorm, CxNormlz
CxPerspective
   Makes a perspective matrix

SYNOPSIS
   #include <code/matrix.h>
   void CxPerspective (CxMatrix mat, double p)

ARGUMENTS
   mat       Perspective matrix to be made
   p         Perspective factor

DESCRIPTION
   The perspective matrix has the following form (row form): p must be greater than CX_EPSILONSS.
   (CX_EPSILONSS is defined in <code/const.h>)

\[
  \text{mat} = \begin{bmatrix}
  1 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & 0 & 1 & 1/p \\
  0 & 0 & 0 & 1 \\
  \end{bmatrix}
\]

EXAMPLE

   CxPerspective ( mat, 2 ) ;
**CxPlane**

Finds the minimum distance from a plane to the coordinate origin

**SYNOPSIS**

```c
#include <code/matrix.h>
void CxPlane (CxVector point, CxVector normal, double *d)
```

**ARGUMENTS**

- `point`: A point in the plane
- `normal`: The plane normal vector
- `d`: Minimum distance from the plane to the coordinate origin

**DESCRIPTION**

This function finds the minimum distance from a given plane to the coordinate origin. Through this function the plane normal is always set to point away from the coordinate origin.

**EXAMPLE**

The following code segment determines whether a given plane is within the given sensitivity distance with a specified tolerance `TOL`.

```c
#define TOL 0.00001
CxVector point, normal;
double min_dis, sensitive_dis=1.0;
/* load parameters for a plane */
point[0] = 5.0;
point[1] = 3.0;
point[2] = 4.0;
normal[0] = 1.0;
normal[1] = normal[2] = 0.0;
/* determine minimum distance */
CxPlane(point, normal, &min_dis);
/* check minimum distance */
if (fabs (min_dis - sensitive_dis) < TOL) {
    printf ("The given plane is within the sensitive distance\n");
} else {
    printf ("The given plane is not within the sensitive distance\n");
}
```
CxPrintMxn

Prints an m \times n matrix to stdout

SYNOPSIS

#include <code/matrix.h>
void CxPrintMxn (double *mat_array, long nrow, long ncol)

ARGUMENTS

mat_array Matrix to be printed
nrow Number of rows in the matrix
ncol Number of columns in the matrix

DESCRIPTION

This function prints an m by n matrix to stdout. The matrix passed as mat_array contains nrow times ncol elements.

EXAMPLE

The following code makes a 2x3 matrix, and prints it to stdout.

double *mat_array;

/* allocate memory */
mat_array = (double *) calloc (6, sizeof (double));

/* load the matrix array */
mat_array[0] = 1.0;
mat_array[1] = 2.0;
mat_array[2] = 3.0;
mat_array[3] = 4.0;
mat_array[4] = 5.0;
mat_array[5] = 6.0;

CxPrintMxn (mat_array, 2, 3);

The 2x3 matrix will be printed on screen in following form:

\[
\begin{bmatrix}
  1.0 & 2.0 & 3.0 \\
  4.0 & 5.0 & 6.0
\end{bmatrix}
\]

SEE ALSO

CxPrintm
**CxPrintm**

Prints a 4x4 matrix to stdout

**SYNOPSIS**

```c
#include <code/matrix.h>
void CxPrintm(CxMatrix mat)
```

**ARGUMENTS**

mat Matrix to be printed

**DESCRIPTION**

The function `CxPrintm` is used specifically to print 4x4 matrices such as pose matrices, and is useful for debugging. The matrix components are printed on standard output in the following (column) form:

```
(0,0)  (0,1)  (0,2)  (0,3)
(1,0)  (1,1)  (1,2)  (1,3)
(2,0)  (2,1)  (2,2)  (2,3)
(3,0)  (3,1)  (3,2)  (3,3)
```

**EXAMPLE**

```
CxMatrix mat={ {4.0, 3.0, 2.0, 1.0 },
               {1.1, 2.0, 5.0, 3.0},
               {4.0, 7.0, 21.0, 15.0},
               {0.0, 0.0, 0.0, 1.0} } ;

CxPrintm ( mat ) ;
```

The following output appears on the screen:

```
4.000000  3.000000  2.000000  1.000000
1.100000  2.000000  5.000000  3.000000
4.000000  7.000000 21.000000 15.000000
0.000000  0.000000  0.000000  1.000000
```

**SEE ALSO**

`CxPrintMxn`
CxPrintv
Prints a vector to stdout

SYNOPSIS
#include <code/matrix.h>
void CxPrintv(CxVector vec)

ARGUMENTS
vec Vector to be printed

DESCRIPTION
This function prints any vector to standard output and is useful for debugging.

EXAMPLE

CxVector vec= {2.0, 7.0, 8.0} ;

CxPrintv ( vec ) ;
The following is displayed:
2.000000 7.000000 8.000000

SEE ALSO
CxPrintMxn
**CxPuttr**

Puts a translation vector in the specified pose matrix

**SYNOPSIS**

```c
#include <code/matrix.h>
void CxPuttr(CxMatrix mat, CxVector vec, CxMatrix mat_out)
```

**ARGUMENTS**

- **mat** Pose matrix whose rotational submatrix is to be used
- **vec** Translational vector to be put in the output pose matrix
- **mat_out** Output pose matrix

**DESCRIPTION**

This function assigns a 4x4 matrix \texttt{mat\_out} to be a duplicate of the input matrix \texttt{mat} except that the position values are replaced by those of the input vector. In other words, the new matrix will have the same rotational submatrix as the original matrix, but will use the vector for its positional components.

**EXAMPLE**

The following example code creates a pose matrix, which is, rotated about its Y axis 90 degrees, and displaced at (10, 30, 15) relative to the input matrix.

```c
CxMatrix mat1, mat2;
CxVector vec;

/* create mat1 as rotated 90 degrees about Y axis */
CxMakero (mat1, 1.5708, 'Y');

/* assign values to vec */
vec[0] = 10.0;
vec[1] = 30.0;
vec[2] = 15.0;

/* put vec in mat1 to make mat2 */
CxPuttr(mat1, vec, mat2)

/* print out mat2 to stdout */
CxPrintm (mat2);
```

**SEE ALSO**

- CxMaketr
**CxReadMat**

Reads a $4 \times 4$ matrix from a file

**SYNOPSIS**

```c
#include <stdio.h>
#include <code/matrix.h>
void CxReadMat (CxMatrix mat, FILE *fileId)
```

**ARGUMENTS**

- `mat` : Matrix into which resulting data will be stored
- `fileId` : File descriptor of previously opened data file

**DESCRIPTION**

This function is used to read a $4 \times 4$ matrix from a previously opened data file. The file descriptor must be obtained using the `fopen` system call.

**RETURN VALUE**

None

**EXAMPLE**

In this example, the file pointer `data_file` points to a file containing the following data:

The example code segment will read the data into a $4 \times 4$ matrix, and then print that matrix to `stdout`.

```c
FILE *data_file;
CxMatrix new_mat;
.
CxReadMat(new_mat, data_file);
CxPrintm (new_mat);
```

**SEE ALSO**

- `CxWriteMat`
CXRlmat

Finds the relative matrix that relates two matrices

SYNOPSIS
#include <code/matrix.h>
void CXRlmat(CxMatrix first_mat, CxMatrix second_mat, CxMatrix rel_mat)

ARGUMENTS
first_mat  First pose matrix
second_mat Second pose matrix
rel_mat    Returns a relative matrix that describes the first matrix with respect to the second matrix

DESCRIPTION
This function finds the pose matrix relating two input pose matrices. Note that, in order to return useful information, especially when using rel_mat to find the relative poses of CODE frames, both first_mat and second_mat must be relative to the same coordinate frame.

The second_mat describes the transformation from frame s to w. The first_mat describes the transformation from frame f to w. The rel_mat return from this function describes the transformation from frame f to s.

EXAMPLE
The following program generates a transformation matrix for a point which is at position (1.0, 5.0, 10.0) and having a frame rotated 180.0 degrees about Z axis with respect to the main reference. Then it generates a transformation matrix for another point, which is at position (2.0, 6.0, 11.0) and whose frame is rotated about Z axis by 90.0 degrees with respect to the main reference frame. Then the program computes and prints out the relative transformation matrix as shown below for the point 1 with respect to the point 2. Relative matrix:

\[
\begin{bmatrix}
0.0 & 1.0 & 0.0 & 0.0 \\
-1.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 1.0 & 0.0 \\
-1.0 & 1.0 & -1.0 & 1.0 \\
\end{bmatrix}
\]
The matrix shows that point 1 is at (-1.0, 1.0, 1.0) and is rotated 90.0 degrees along Z axis with respect to the frame at point 2.

```c
void print_matrix (CxMatrix mat, char *mat_name);
#include <stdio.h>
#include <code/matrix.h>

void main(void)
{
    CxMatrix  trans_pt1, trans_pt2;
    CxVector  position, angle;
    CxMatrix  rel_mat;

    position[0] = 1.0; position[1] = 5.0; position[2] = 10.0;
    angle[0] = 0.0; angle[1] = 0.0; angle[2] = CX_PI;
    CxXyzmat ("XYZ", angle, position, trans_pt1);
    printf( " Point 1: \n" );
    CxPrintm( trans_pt1 );

    position[0] = 2.0; position[1] = 6.0; position[2] = 11.0;
    angle[0] = 0.0; angle[1] = 0.0; angle[2] = CX_PI/2;
    CxXyzmat ("XYZ", angle, position, trans_pt2);
    printf( " Point 2
" );
    CxPrintm( trans_pt2 );

    CxRlmat (trans_pt1, trans_pt2, rel_mat);
    printf( " Relative matrix: \n" );
    CxPrintm(rel_mat);
}

SEE ALSO
CxCompareMat
```
CxRmul4x4

Multiplies two 3 × 3 rotational submatrices of two 4 × 4 matrices

SYNOPSIS

#include <code/matrix.h>

void CxRmul4x4 (CxMatrix mat1, CxMatrix mat2, CxMatrix mat_result)

ARGUMENTS

mat1  First matrix in the multiplication operation
mat2  Second matrix in the operation
mat_result  Returns the matrix resulting from the multiplication operation

DESCRIPTION

The function multiplies two 3 × 3 rotational submatrices of the given 4 × 4 matrices. If R is used to represent the 3 × 3 rotational submatrix of a 4 × 4 pose matrix, then R_result = R_1 * R_2. The resulting matrix, mat_result, uses the same displacement vector as that of the second matrix or mat2.

EXAMPLE

The following example generates two transformation matrices and then uses the CxRmul4x4 rmul4 × 4 function to multiply two 3 × 3 submatrices. Observe that the resultant matrix has the same position vector as that of the second matrix.

Matrix 1:

\[
\begin{bmatrix}
-1.0 & 0.0 & 0.0 & 0.0 \\
0.0 & -1.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 1.0 & 0.0 \\
1.0 & 5.0 & 10.0 & 1.0 \\
\end{bmatrix}
\]

Matrix 2:

\[
\begin{bmatrix}
0.0 & 1.0 & 0.0 & 0.0 \\
-1.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 1.0 & 0.0 \\
10.0 & 50.0 & 100.0 & 1.0 \\
\end{bmatrix}
\]

Resultant Matrix:

\[
\begin{bmatrix}
0.0 & -1.0 & 0.0 & 0.0 \\
1.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 1.0 & 0.0 \\
10.0 & 50.0 & 100.0 & 1.0 \\
\end{bmatrix}
\]
#include <stdio.h>
#include <code/const.h>
#include <code/matrix.h>

void main(void)
{
    CxMatrix mat1, mat2;
    CxVector position, angle;
    CxMatrix final_mat;

    position[0] = 1.0; position[1] = 5.0; position[2] = 10.0;
    angle[0] = 0.0; angle[1] = 0.0; angle[2] = CX_PI;
    CxXyzmat("XYZ", angle, position, mat1);
    printf(" Point 1:\n");
    CxPrintm(mat1);

    position[0] = 10.0; position[1] = 50.0; position[2] = 100.0;
    angle[0] = 0.0; angle[1] = 0.0; angle[2] = CX_PI/2;
    CxXyzmat("XYZ", angle, position, mat2);
    printf(" Point 2:\n");
    CxPrintm(mat2);

    CxRmul4x4(mat1, mat2, final_mat);
    printf(" Resultant Matrix:\n");
    CxPrintm(final_mat);
}

SEE ALSO

CxMul4x4,CxMultMxn
**CxRolyaw**

Extracts ordered Yaw, Pitch, and Roll angles (Z Y X) from a pose matrix

**SYNOPSIS**

```
#include <code/matrix.h>
void CxRolyaw (CxMatrix mat, CxVector angles)
```

**ARGUMENTS**

- **mat** Pose matrix to be analyzed
- **angles** Returns the yaw, pitch, and roll angles (Z Y X) in radians of the specified pose matrix

**DESCRIPTION**

This function takes a pose matrix as input and returns yaw, pitch, and roll angles representing the rotational portion of the pose matrix.

This set of three angles is a convenient and compact way to represent the same information contained in the rotational portion of a pose matrix. Some robots require the rotational information to be in this form. The “Yaw-Pitch-Roll” convention describes a rotation first about the principal Z axis followed by a rotation about the principal Y axis and then a final rotation about the principal X axis. Similar representations exist for the rotation of a pose matrix. Each considers the rotations to have occurred about a different set of axes.

**EXAMPLE**

The following program generates a transformation matrix by first rotating about X axis by 90.0 degrees and then rotating about resultant Z axis by 90.0 degrees. Then it computes Yaw, Pitch and Roll for that transformation matrix and prints out their values as follows: Yaw: 0.000000 Pitch: -90.000000 Roll: 90.000000. It can be seen from the results that the same transformation can be achieved by rotating about the principal Y axis by -90.0 degrees followed by rotating about the principal X axis by 90.0 degrees.

```c
#include <stdio.h>
#include <code/matrix.h>

void main(void)
{
    CxMatrix new_axis_mat;
    CxVector angle, position;
    CxVector yrp;

    position[0] = 1.0; position[1] = 5.0; position[2] = 10.0;
    angle[0] = CX_PI/2.0; angle[1] = 0.0; angle[2] = CX_PI/2.0;
    CxXyzmat ("XYZ", angle, position, new_axis_mat);

    CxRolyaw (new_axis_mat, yrp);
    printf (" Yaw: %lf Pitch: %lf Roll: %lf\n",
            yrp[0]*180.0/CX_PI, yrp[1]*180.0/CX_PI, yrp[2]*180.0/CX_PI);
}
```

**SEE ALSO**

CxEuler, CxXyzmat
**CxRot**

Calculates the rotational elements of a pose matrix given a rotational order and angles

**SYNOPSIS**

```c
#include <code/matrix.h>
long CxRot (char order[4], CxVector angles, CxMatrix rot_submat)
```

**ARGUMENTS**

- `order` Specifies the order of the axes
- `angles` Specifies the angles in radians to rotate the pose
- `rot_submat` Returns the rotational sub-matrix of a standard pose matrix

**DESCRIPTION**

This function takes as input a set of axes and angles, and returns a 4 × 4 matrix in which the 3 × 3 rotational submatrix has been calculated.

The `order` character array contains the ordered names of the axes to be rotated about. The possible choices as axis names are the principal axes X, Y, and Z. The `angles` vector contains the corresponding angles in radians. The resulting matrix has the following form:

\[
\begin{bmatrix}
R & 0 \\
0^T & 1
\end{bmatrix}
\]

where

- `R` is the resulting 3 × 3 rotation submatrix
- `0` is a zero vector

**RETURN VALUES**

This function returns 0, if successful; otherwise, -1 (CX_ERROR) is returned.

**EXAMPLE**

The following code defines a character array `ord` specifying the order of the axes (XZY in this case) and a vector `angs` specifying the angles to rotate about each axis, and then calls `CxRot` to calculate the 3x3 rotation submatrix `rotmat`.

```c
#define DEG_TO_RAD 1.74532925e-2
CxVector angs;
char ord[4];
CxMatrix rotmat;

/* Parameters for axis order and revolution angles */
ord[0] = 'X';
ord[1] = 'Z';
ord[2] = 'Y';
ord[3] = 0;
```
angs[0] = 10.0 * DEG_TO_RAD; /* angles in degrees */
angs[1] = 25.0 * DEG_TO_RAD;
angs[2] = 15.0 * DEG_TO_RAD;

/* Call function to get rotational matrix */
CxRot (ord, angs, rotmat)

SEE ALSO

CxMakero, CxRotm
**CxRotm**

Makes a pose matrix from a screw angle, screw vector, and translation vector

**SYNOPSIS**

```c
#include <code/matrix.h>
void CxRotm (double phi, CxVector screw_vec, CxVector trans_vec, CxMatrix mat)
```

**ARGUMENTS**

- **phi**
  
  Amount of rotation in radians about the screw axis

- **screw_vec**
  
  Direction of the screw axis (normalized)

- **trans_vec**
  
  Translational vector to be incorporated into the pose matrix

- **mat**
  
  Returns the pose matrix with the specified rotation and translation

**DESCRIPTION**

The function `CxRotm` is perhaps the most useful and general of the pose generating functions. This function can effectively replace the `CxMakeRo` function by selecting a screw vector that is one of the principal axes. In any case, the screw vector should be normalized. This makes the screw vector equivalent to the directional cosines of the screw axis (see `CxNormlz`). Note also that `phi` can be assigned zero to make a pure translation matrix.

**EXAMPLE**

The following code defines the various input components to be used in creating a pose matrix (screw angle, screw vector, and translation vector) and then calls `CxRotm` to create the new matrix.

```c
#define DEG_TO_RAD 1.74532925e-2
double rotamt= 45 * DEG_TO_RAD;
CxVector sc_vector, translate;
CxMatrix newmat;

/* Vector defining the direction of the screw axis */
sc_vector[0] = 3.5;
sc_vector[1] = 4.0;
sc_vector[2] = 2.0;

/* Normalize the vector defining the screw axis */
CxNormlz (sc_vector);

/* Vector specifying translation */
translate[0] = 5.27;
translate[1] = 2.53;
translate[2] = 0.41;

/* Call function to make pose matrix named newmat */
CxRotm (rotamt, sc_vector, translate, newmat)
```

**SEE ALSO**

`CxScrew`
CxScale

Makes a $4 \times 4$ scaling matrix

**SYNOPSIS**

```c
#include <code/matrix.h>
void CxScale (CxMatrix mat, CxVector scale_vec)
```

**ARGUMENTS**

- `mat` Returns the scaling matrix
- `scale_vec` Scaling factors

**DESCRIPTION**

The scaling matrix takes on the following form:

$$
\begin{bmatrix}
    sx & 0 & 0 & 0 \\
    0 & sy & 0 & 0 \\
    0 & 0 & sz & 0 \\
    0 & 0 & 0 & 1 \\
\end{bmatrix}
$$

Where $sx=\text{scale}_{vec}[0]$; $sy=\text{scale}_{vec}[1]$; and $sz=\text{scale}_{vec}[2]$. 
**CxScalevec**

Scales a vector by a factor

**SYNOPSIS**

```c
#include <code/matrix.h>
void CxScalevec (CxVector vec, double factor)
```

**ARGUMENTS**

- `vec` Vector to be scaled
- `factor` Scale factor

**DESCRIPTION**

This function is used to scale a vector by a factor.

**EXAMPLE**

The following code scales a vector by a factor of 2.5.

```c
CxVector vec=(1.0, 2.0, 3.0);
/* scale the vector by a factor of 2.5 */
CxScalevec (vec, 2.5);
```

At this point, `vec` will become (2.5, 5.0, 7.5).

**SEE ALSO**

- CxScale
CxScrew

Finds the screw angles and screw vector from a pose matrix

SYNOPSIS

```
#include <code/matrix.h>
void CxScrew (CxMatrix mat, CxVector screw_vector, double *screw_angle)
```

ARGUMENTS

- **mat**: Pose matrix to be analyzed
- **screw_vector**: Normalized screw vector corresponding to minimum screw angles
- **screw_angle**: Minimum screw angle in radians

DESCRIPTION

Given the rotational submatrix of a pose matrix, a screw axis and angle can be determined, which physically represents the rotation of a frame about an axis through the origin of a reference frame. This function returns the screw vector corresponding to the minimum rotation angle necessary to orient the frame represented by the pose matrix. Another solution, the rotation angle \((2\pi - \text{screw\_angle})\) about a screw vector opposite (negative of) **screw\_vector**, is not returned.

EXAMPLE

The following program computes a transformation matrix to position a node at \((1.0, 5.0, 10.0)\) with axes first rotated about X axis by 10.0 degrees then rotated by 20.0 degrees about the resultant Z axis. It then prints out the resultant transformation matrix. Next, it gets the rotation angles using this matrix to verify that the transformation matrix correctly reflects the rotations. It then computes the screw angle and screw vector, which would generate the same transformation matrix. Finally, it computes the transformation matrix using this screw vector and the screw angle, then prints the transformation matrix out. Note that both matrices are identical.

```c
#include <stdio.h>
#include <code/matrix.h>

void main (void)
{
    CxMatrix new_axis_mat;
    CxVector angle, position;
    CxVector rot_angle, screw_vec;
    double screw_angle;

    position[0] = 1.0; position[1] = 5.0; position[2] = 10.0;
    angle[0] = CX_PI*10.0/180.0;
    angle[1] = 0.0;
    angle[2] = CX_PI*20.0/180.0;
    CxXyzmat ("XYZ", angle, position, new_axis_mat);
    printf( "Rotation through Normal way\n");
    CxPrintm( new_axis_mat );
    CxXyzang (new_axis_mat, rot_angle);
    ```
printf("Rotation angles: %lf %lf %lf
",
    rot_angle[0]*180.0/CX_PI,
    rot_angle[1]*180.0/CX_PI,
    rot_angle[2]*180.0/CX_PI);

CxScrew(new_axis_mat, screw_vec, &screw_angle);
printf("Screw angle: %lf\n", screw_angle*180.0/CX_PI);

printf("Screw vector: %lf %lf %lf \n", 
    screw_vec[0], screw_vec[1], screw_vec[2]);

CxRotm(screw_angle, screw_vec, position, new_axis_mat);
printf("Rotation using screw vector\n");
CxPrintm(new_axis_mat);

SEE ALSO
CxRotm
**CxShear**

Makes a shearing matrix

**SYNOPSIS**

```c
#include <code/matrix.h>
void CxShear(CxMatrix mat, CxPose shr)
```

**ARGUMENTS**

- **mat**  Returns the shearing matrix
- **shr**  Specifies the shearing components

The type *CxPose* is defined in *matx_defs.h* as follows:

```c
typedef double CxPose[6];
```

**DESCRIPTION**

The shearing matrix takes on the following form:

```
\[
\begin{bmatrix}
1 & Shyx & Shzx & 0 \\
Shxy & 1 & Shzy & 0 \\
Shxz & Shyz & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
```

where:

- Shyx = shr[0]; Shzx = shr[1];
- Shxy = shr[2]; Shzy = shr[3];
- Shxz = shr[4]; Shyz = shr[5].

**EXAMPLE**

```c
#include <code/matrix.h>

CxMatrix mat;
CxPose *shearing_component={3.0, 40.0, 7.0, 8.0, 9.0, 2.0 };

CxShear ( mat, shearing_component ) ;
```
**CxTransp**

Transposes a matrix

**SYNOPSIS**

```
#include <code/matrix.h>
void CxTransp (CxMatrix mat, CxMatrix trp_mat, long n)
```

**ARGUMENTS**

- **mat**
  - Matrix to be transposed
- **trp_mat**
  - Returns the transpose of the specified matrix
- **n**
  - Dimensions of the matrices

**DESCRIPTION**

When a square matrix is transposed, its rows and columns are interchanged. For example, a pose matrix in column form may be represented as follows:

\[
T_c = \begin{bmatrix}
    nx & ox & ax & px \\
    ny & oy & ay & py \\
    nz & oz & az & pz \\
    0  & 0  & 0  & 1
\end{bmatrix}
\]

The same matrix in row form is written as the transpose of its corresponding column form:

\[
T_r = (T_c)^t = \begin{bmatrix}
    nx & ny & nz & 0 \\
    ox & oy & oz & 0 \\
    ax & ay & az & 0 \\
    px & py & pz & 1
\end{bmatrix}
\]

**NOTE:** Since CxMatrix is defined as a 4×4 pose matrix, dimension of the matrix, n, should always be set to 4.

**EXAMPLE**

```
CxMatrix tmat, mat={{4.0, 3.0, 2.0, 1.0},
                   {1.1, 2.0, 5.0, 3.0},
                   {4.0, 7.0, 21.0, 15.0},
                   {0.0, 0.0, 0.0, 1.0} };

CxTransp ( mat, tmat, 4 ) ;
```
**CxVecad**

Adds one vector with a second scaled vector

**SYNOPSIS**

```c
#include <code/matrix.h>
void CxVecad (double constant, CxVector vec1, CxVector vec2, CxVector result_vec)
```

**ARGUMENTS**

- **constant**  
  Constant (scaling factor) by which the second vector is multiplied
- **vec1**  
  First vector
- **vec2**  
  Second vector
- **result_vec**  
  Returns the result of the vector addition

**DESCRIPTION**

The result vector is calculated as follows:

```
result_vec = vec1 + constant * vec2
```

**EXAMPLE**

The following program adds two vectors, then prints out the result. The resultant vector is: [2.0 4.0 10.0]

```c
#include <stdio.h>
#include <code/matrix.h>

void main()
{
    CxVector vec1, vec2;
    CxVector result;
    vec1[0] = 1.0; vec1[1] = 2.0; vec1[2] = 5.0;
    vec2[0] = 10.0; vec2[1] = 20.0; vec2[2] = 50.0;
    CxVecad (0.1, vec1, vec2, result);
    printf ("Resultant vector is: %lf %lf %lf \n", result[0], result[1], result[2]);
}
```

**SEE ALSO**

CxVeceql, CxVecmlt, CxVecneg, CxVecsub
**CxVeceql**

Equates two vectors making vec2 = vec1

**SYNOPSIS**

```c
#include <code/matrix.h>
void CxVeceql (CxVector vec1, CxVector vec2)
```

**DESCRIPTION**

vec1  Vector to be duplicated
vec2  Returns a duplication of the specified vector

**ARGUMENTS**

Equating vec2 to vec1 is the same as performing the following three assign statements:

```c
vec2[0]  =  vec1[0];
vec2[1]  =  vec1[1];
vec2[2]  =  vec1[2];
```

**EXAMPLE**

```c
CxVector vec2, vec1={1.0, 3.0, 5.0};
.
.
CxVeceql ( vec1, vec2 ) ;
```

**SEE ALSO**

CxVecad, CxVecmlt, CxVecneg, CxVecsub
**CxVecmlt**

Transforms a vector through the full pose matrix or through the rotational portion only

**SYNOPSIS**

```c
#include <code/matrix.h>
void CxVecmlt (long rotation_only, CxMatrix mat, CxVector in_vec, CxVector out_vec)
```

**ARGUMENTS**

- `rotation_only` Transformation option (CX_TRUE for rotation only; CX_FALSE for full pose matrix). CX_TRUE and CX_FALSE are defined in `<code/const.h>`
- `mat` Pose matrix to be used in the vector transformation
- `in_vec` Vector to be transformed through the pose matrix
- `out_vec` Returns the transformed vector

**DESCRIPTION**

The following equations show the arrangement used to transform a vector, first through an entire pose matrix, then through the rotational portion only.

Full transform (parameter `rotation_only` set to CX_FALSE):

\[
\begin{bmatrix}
\text{out}_\text{vec}[0] \\
\text{out}_\text{vec}[1] \\
\text{out}_\text{vec}[2] \\
1
\end{bmatrix}
= \begin{bmatrix}
\text{in}_\text{vec}[0] \\
\text{in}_\text{vec}[1] \\
\text{in}_\text{vec}[2] \\
1
\end{bmatrix}
\begin{bmatrix}
r_{11} & r_{12} & r_{13} & 0 \\
r_{21} & r_{22} & r_{23} & 0 \\
r_{31} & r_{32} & r_{33} & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Rotational only (parameter `rotation_only` set to CX_TRUE):

\[
\begin{bmatrix}
\text{out}_\text{vec}[0] \\
\text{out}_\text{vec}[1] \\
\text{out}_\text{vec}[2] \\
1
\end{bmatrix}
= \begin{bmatrix}
\text{in}_\text{vec}[0] \\
\text{in}_\text{vec}[1] \\
\text{in}_\text{vec}[2] \\
1
\end{bmatrix}
\begin{bmatrix}
r_{11} & r_{12} & r_{13} & 0 \\
r_{21} & r_{22} & r_{23} & 0 \\
r_{31} & r_{32} & r_{33} & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

**EXAMPLE**

This program computes the new coordinates for a point assuming coordinate axes are moved to a new position (1.0, 5.0, 10.0) and rotated at 180.0 degrees about the x-axis and 180.0 degrees about the z-axis with respect to the old axes. It first computes the new coordinates without taking into account the translations and prints them out. Then it takes into account rotation and translation and computes the new coordinates and prints them out.

New coordinates without translation effect: [-1.0 1.0 -1.0]

New coordinates with translation effect: [0.0 6.0  9.0]

#include `<stdio.h>`

#include <code/matrix.h>

void main (void)
{
    CxMatrix new_axis_mat;
    CxVector old_coord, new_coord;
    CxVector angle, position;

    position[0] = 1.0; position[1] = 5.0; position[2] = 10.0;
    angle[0] = CX_PI; angle[1] = 0.0; angle[2] = CX_PI;
    CxXyzmat ("XYZ", angle, position, new_axis_mat);

    old_coord[0] = 1.0, old_coord[1] = 1.0; old_coord[2] = 1.0;
    CxVecmlt (CX_TRUE, new_axis_mat, old_coord, new_coord);
    printf ("New Coordinates are: %lf %lf %lf \n", \ 
            new_coord[0], new_coord[1], new_coord[2]);

    CxVecmlt (CX_FALSE, new_axis_mat, old_coord, new_coord);
    printf ("New Coordinates are: %lf %lf %lf \n", \ 
            new_coord[0], new_coord[1], new_coord[2]);
}

SEE ALSO

CxVecad, CxVeceql, CxVecneg, CxVecsub
CxVecneg

Negates a vector

SYNOPSIS
#include <code/matrix.h>
void CxVecneg (CxVector vec_in, CxVector vec_out)

ARGUMENTS
vec_in Vector to be negated
vec_out Returns the negated vector

DESCRIPTION
This function returns the negative of a vector (vec_out = – vec_in).

EXAMPLE
CxVector nvec, vec={1.0, 2.0, 3.0} ;
CxVecneg ( vec, nvec ) ;

SEE ALSO
CxVecad, CxVeceql, CxVecmlt, CxVecsub
**CxVecsub**

Subtracts a vector from another

**SYNOPSIS**

```c
#include <code/matrix.h>
void CxVecsub (CxVector vec1, CxVector vec2, CxVector vec_result)
```

**ARGUMENTS**

- vec1 Vector to be subtracted from
- vec2 Vector to subtract
- vec_result Contains the result of vec1 - vec2

**DESCRIPTION**

The resultant vector is computed as follows:

```c
vec_result = vec1 - vec2
```

**EXAMPLE**

```c
CxVector vec1={1.0, 2.0, 4.0}, vec2={5.0, 4.0, 3.0};
CxVector rvec;
CxVecsub ( vec1, vec2, rvec ) ;
printf ( "%lf %lf %lf\n", rvec[0], rvec[1], rvec[2] ) ;
```

**SEE ALSO**

CxVecad, CxVeceq, CxVecmult, CxVecneg
**CxWriteMat**

Writes a $4 \times 4$ matrix to a file

**SYNOPSIS**

```c
#include <stdio.h>
#include <code/matrix.h>
void CxWriteMat (CxMatrix mat ,FILE *fp)
```

**ARGUMENTS**

- `mat` Matrix to write file
- `fp` File pointer

**DESCRIPTION**

This function writes a $4 \times 4$ matrix to a file specified by `fp`. The matrix is written row by row, from top to bottom. Within each row, the matrix is written from left to right.
**CxxYxZang**

Extracts equivalent ordered angles (X Y Z) for the rotation of a pose matrix

**SYNOPSIS**

```c
#include <code/matrix.h>
void CxXyzang (CxMatrix mat, CxVector angles)
```

**ARGUMENTS**

- **mat**: Pose matrix to be analyzed
- **angles**: Returns the ordered angles (X Y Z) that are equivalent to the rotation of the specified pose matrix

**DESCRIPTION**

This set of three angles is a convenient and compact way to represent the information contained in the rotational portion of a pose matrix. Some robots require the rotational information to be in this form. This angle convention describes a rotation first about the X axis followed by a rotation about the principal Y axis and then a final rotation about the principal Z axis. Similar representations exist for the rotation of a pose matrix. Each considers the rotations to have occurred about a different set of axes.

**EXAMPLE**

The following program computes a transformation matrix to position a node at (1.0, 5.0, 10.0) with axes first rotated about x-axis by 10.0 degrees, then by 20.0 degrees about the resultant z-axis. It then prints out the resultant transformation matrix. Next, it gets the rotation angles using this matrix to verify that the transformation matrix correctly reflects the rotations. It then computes the screw angle and screw vector, which would generate the same transformation matrix. Finally, it computes the transformation matrix using this screw vector and the screw angle, then prints the transformation matrix out. Note that both matrices are identical.

```c
#include <stdio.h>
#include <code/matrix.h>

void main (void)
{
    CxMatrix new_axis_mat;
    CxVector angle, position;
    CxVector rot_angle, screw_vec;
    double screw_angle;

    position[0] = 1.0; position[1] = 5.0; position[2] = 10.0;
    angle[0] = CX_PI*10.0/180.0;
    angle[1] = 0.0;
    angle[2] = CX_PI*20.0/180;
    CxxYxZmat ("XYZ", angle, position, new_axis_mat);
    printf( "Rotation through Normal way\n");
    CxPrintm( new_axis_mat );

    CxXyzang (new_axis_mat, rot_angle);
    printf ("Rotation angles: %lf %lf %lf \n",
```
rot_angle[0]*180.0/CX_PI,
rot_angle[1]*180.0/CX_PI,
rot_angle[2]*180.0/CX_PI);

CxScrew (new_axis_mat, screw_vec, &screw_angle);
printf("Screw angle: %lf\n", screw_angle*180.0/CX_PI);
printf("Screw vector: %lf %lf %lf \n", 
    screw_vec[0], screw_vec[1], screw_vec [2]);

CxRotm (screw_angle, screw_vec, position, new_axis_mat);
printf("Rotation using screw vector\n");
CxPrintm(new_axis_mat);
}

SEE ALSO

CxEuler, CxRolyaw, CxXyzmat
CxXyzmat

Makes a pose matrix from three angles, the corresponding axes, and a translation vector

SYNOPSIS

#include <code/matrix.h>
long CxXyzmat (char axes[4], CxVector angles, CxVector trans_vec, CxMatrix mat)

ARGUMENTS

axes       Principal axes about which the pose matrix is rotated
angles     Angles in radians corresponding to the specified axes about which the pose matrix is to be rotated
trans_vec  Translation to be incorporated into the pose matrix
mat        Returns the pose matrix with the calculated rotation sub-matrix and the specified translation

DESCRIPTION

This function might be useful where data already exists in the more compact angle-axes/translation form and has to be converted into the matrix form to be usable by some other system.

RETURN VALUE

This function returns the value of the specified signal if successful; otherwise, -1 (CX_ERROR) is returned.

EXAMPLE

The following code segment extracts Euler angles and translation vector from a pose matrix and makes another pose matrix with the extracted Euler angles and translation vector using the function CxXyzmat. It then compares the rotational submatrices of the original and the new pose matrices to check if they are the same (the comparison is made using CxCompareMat, also in the “Matrix” library).

CxMatrix oldmat, newmat, relmat;
char axes[4];
CxVector tvec, angs;

/* get original matrix oldmat */
.
/* extracts euler angles */
CxEuler (oldmat, angs);
CxFindtr (oldmat, tvec);

/* make new pose matrix */
strcpy (axes, "ZYZ");
CxXyzmat (axes, angs, tvec, newmat);

/* compare the rotational portion of old and new matrices */
if (CxCompareMat (oldmat, newmat, relmat, 0.1, 0.01, CX_SCREW_ONLY)){
printf ("Conversion Successful...\n");
}
else {
    printf ("Conversion failed...\n");
    printf ("Please check your compare tolerances.\n");
}

SEE ALSO

CxEuler, CxMakero, CxRolyaw, CxXyzang
PMAC - Specific Functions
**CxConvertGatherData**

Converts the upload samples from PMAC format to doubles

**SYNOPSIS**

```
#include <code/robpac.h>
long CxConvertGatherData (char *buffer, char *format, double *data,
                         long *data_size, long *err_no)
```

**ARGUMENTS**

- **buffer**  Buffer that stores gathered samples
- **format**  Information that is returned by CxPmacGatherInit
- **data**    Return the converted samples
- **data_size**  Number of sample sets that are converted
- **err_no**  Error number if an error occurs

**DESCRIPTION**

This function converts the gathered samples from PMAC format to the double type. The format string is obtained in the CxPmacGatherInit call, and contains information on how to convert the gathered samples into an array of doubles. The user should allocate memory for data array of size `num_sig` (the number of signals to be gathered as specified in CxPmacGatherInit) times `actual_sample` (the actual number of sample sets that are uploaded is returned from CxPmacGatherUpload). `data_size` is returned from the CxConvertGatherData call and its value is the number of sample sets that are converted.

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAC_DATA_FORMAT_ERROR</td>
<td>The content of the buffer does not match the information contained in format string</td>
</tr>
</tbody>
</table>

**EXAMPLE**

This example demonstrates how the parameters for CxConvertGatherData should be set up.

```c
char *buffer;
char format[10];
double *data;
long num_sig, actual_sample, data_size, err_no;

/* buffer has been allocated for CxPmacGatherUpload call */

data = ( double * ) calloc ( num_sig*actual_sample, sizeof(double));
```
CxConvertGatherData ( buffer, format, data, &data_size, &err_no);

SEE ALSO
CxPmacGatherInit, CxPmacGatherStart, CxPmacGatherStop, CxPmacGatherUpload
### CxPmacGatherInit

Initializes PMAC data gathering operation

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxPmacGatherInit (CxController cntrl, long num_sig, long *signals, long period, long mode, long *num_sample, long *buff_size, char *format)
```

**ARGUMENTS**

- `cntrl` Controller ID
- `num_sig` Number of signals to be gathered
- `signals` Array of signals to be gathered
- `period` How often (in PMAC servo cycles) the signals will be gathered
- `mode` Mode of gathering operation, valid options are 0 and 1
- `num_sample` Number of samples to be gathered; valid input is any non-negative number
- `buff_size` Buffer size that is required to upload the gathered samples (in bytes)
- `format` Information for converting the gathered samples

**DESCRIPTION**

This function initializes the parameters for data gathering operation on the PMAC card. The actual data gathering operation will not start until the function `CxPmacGatherStart` is called.

The function `CxGetControllerFromSignal`, in the Process and Mechanism Library, must be called to obtain a valid `cntrl` instance before `CxPmacGatherInit` can be called. The `signals` array contains the signal numbers of the signals that will be gathered. `mode` specifies the mode of data gathering operation; currently 0 is the only option, which gathers the sample points into the PMAC card with no wraparound. On input, if `num_sample` is set to 0, then all the available PMAC memory will be used to store the gathered samples and `num_sample` will be set to the number of sample sets that will be gathered. If `num_sample` is a positive number on input, then it is the number of sample sets to be gathered if there is enough memory; otherwise, `num_sample` will be set to the maximum available sample sets. `buff_size` is returned to the user to allocate a character array, which will be used to upload the gathered samples after data gathering is completed. The user should allocate memory of `format` array before using it. Its size should be at least five times `num_sig`.

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the `CxGetErrorNumber` function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to the CIMServer.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from the CIMServer.</td>
</tr>
<tr>
<td>Error Codes</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-----------------------------------------------------------</td>
</tr>
<tr>
<td>CX_NON_EXISTENT_SIGNAL</td>
<td>The signal number is not defined in the CIMServer’s signal table.</td>
</tr>
<tr>
<td>OAC_NOT_INITIALIZED</td>
<td>Device has not been initialized.</td>
</tr>
<tr>
<td>OAC_GATHER_BUFF_ERROR</td>
<td>Error in PMAC gather buffer.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following example demonstrates initialization of the data gathering process. This example will gather two signals, SIG_2 and SIG_5, every three PMAC servo cycles. In this example num_sample is set to 0, signifying that the user wants to gather as many samples as possible.

```c
Controller Sig_cntrl;
long num_sig, signals[2], period, mode, num_sample, buff_size;
char format[10]; /* the size of format is 2*5 */

if ( ( Sig_cntrl = CxGetControllerFromSignal ( SIG_2 ) ) == CX_NULL ) {
    fprintf(stderr,"can not get controller for given signal\n");
    exit ( -1 );
}

num_sig = 2;
signals[0] = SIG_2;
signals[1] = SIG_5;
period = 3;
num_sample = 0;
mode = 0;

CxPmacGatherInit ( Sig_cntrl, num_sig, signals, period, mode,
    &num_sample, &buff_size, format );

/* CxPmacGatherInit will return the values for buff_size and format which will be used later for uploading and data converting */
```

**SEE ALSO**

CxPmacGatherStart, CxPmacGatherStop, CxPmacGatherUpload, CxConvertGatherData
**CxPmacGatherStart**

Starts the data gathering operation on PMAC card

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxPmacGatherStart (CxController cntrl, long trigger)
```

**ARGUMENTS**

- **cntrl**  Controller ID
- **trigger**  Flag to trigger the data gathering operation; valid options are 0 and 1.

**DESCRIPTION**

This function triggers the data gathering operation on the PMAC card. CxPmacGatherInit must be called before CxPmacGatherStart is called. trigger is a flag that is used to indicate when the data gathering operation will start. (If trigger is 0, the data gathering will start immediately. Otherwise, it will start as soon as input signal 10 of the standard inputs becomes true.)

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to the CIMServer.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from the CIMServer.</td>
</tr>
<tr>
<td>OAC_DEV_CMD_ERROR</td>
<td>Data gathering operation has not been initialized yet.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following example starts the data gathering immediately after CxPmacGatherStart is called.

```c
CxController Sig_cntrl;
long num_sig, signals[2], period, mode, num_sample, buff_size;
char format[10]; /* the size of format is 2*5 */
long trigger=0;

/* CxGetControllerFromSignal should have been called prior to this segment of code. */
CxPmacGatherInit ( Sig_cntrl, num_sig, signals, period, mode,
                   &num_sample, &buff_size, format );
```
trigger = 0;
CxPmacGatherStart ( Sig_cntrl, trigger );

SEE ALSO

CxPmacGatherInit, CxPmacGatherStop, CxPmacGatherUpload, CxConvertGatherData
**CxPmacGatherStop**

Stops the data gathering process

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxPmacGatherStop (CxController cntrl)
```

**ARGUMENTS**

- `cntrl`   Controller ID

**DESCRIPTION**

This function stops the PMAC data gathering operation. It should be called after the data gathering operation has started. The gathered samples are stored in the PMAC memory and can be uploaded by calling `CxPmacGatherUpload`.

**RETURN VALUES**

This function returns **0** if successful; otherwise, **-1 (CX_ERROR)** is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the `CxGetErrorNumber` function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to the CIMServer.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from the CIMServer.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

This example shows how to stop a data gathering process.

```c
CxController Sig_cntrl;
long trigger=0;

CxPmacGatherStart ( Sig_cntrl, trigger ) /* start data gathering */

/* do something here, e.g., send a move to server */

CxPmacGatherStop ( Sig_cntrl );
```
SEE ALSO

CxPmacGatherInit, CxPmacGatherStart, CxPmacGatherUpload, CxConvertGatherData
**CxPmacGatherUpload**

Uploads the gathered samples

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxPmacGatherUpload (CxController cntrl, long start_sample,
                         long num_sample, long *actual_sample, long buff_size,
                         char *buffer)
```

**ARGUMENTS**

- **cntrl**  Controller ID
- **start_sample**  Beginning number of the sample set to upload
- **num_sample**  Number of samples sets to upload
- **actual_sample**  Actual number of sample sets that have been uploaded
- **buff_size**  Size of the storage buffer (in bytes)
- **buffer**  Buffer to store the gathered samples

**DESCRIPTION**

This function uploads the gathered samples from PMAC's memory to the user supplied array, `buffer`. The `start_sample` input parameter indicates the starting number of the sample set to be uploaded, while the `num_sample` parameter indicates the number of sample sets that will be uploaded. If `num_sample` is zero, this function will upload until it reaches the end of the gathered buffer. The `actual_sample` output parameter returns the number of sample sets that have actually been uploaded.

The user must allocate memory for `buffer`, making sure that it contains at least `buff_size` bytes, where `buff_size` is determined using `CxPmacGatherInit`.

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the `CxGetErrorNumber` function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to the CIMServer.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from the CIMServer.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

This example shows how to upload the gathered samples after the data gathering process is terminated.

```c
CxController Sig_cntrl;
long start_sample, num_sample, actual_sample, buff_size;
```
char *buffer;

start_sample = 1;
num_sample = 0;

/* buff_size is returned from CxPmacGatherInit call */
buffer = (char *) malloc ( buff_size * sizeof(char) );

CxPmacGatherUpload ( Sig_cntrl, start_sample, num_sample,
    actual_sample, buff_size, buffer );

SEE ALSO

CxPmacGatherInit, CxPmacGatherStart, CxPmacGatherStop, CxConvertGatherData
CxPmacGetIvar

Gets the current value of the given PMAC i-variable

SYNOPSIS

```c
#include <code/robpac.h>
long CxPmacGetPmacIvar (CxController cntrl, long ivar, double *value)
```

ARGUMENTS

- `cntrl` Controller ID
- `ivar` i-variable number
- `value` Return the current of the given I-variable

DESCRIPTION

This function obtains the current value of the given i-variable (initialization variable). A valid controller ID pointer should be obtained by calling `CxGetControllerFromMech` (see the Controller Management section of the CODE API Programmer’s Reference Manual – Volume 1 for more details) before this function is called. Please refer to PMAC User's Manual and PMAC Software Reference for definitions of i-variables and their respective units.

RETURN VALUES

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS

If the function returns an error condition, the error code can be obtained by using the `CxGetErrorNumber` function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to the CIMServer.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from the CIMServer.</td>
</tr>
</tbody>
</table>

EXAMPLE

This example shows how to get a PMAC i-variable. This code segment will get the value of I130.

```c

.C.

CxController Mech_cntrl;
long ivar;
double ivalue;
CxMechanism my_mech;

Mech_cntrl = CxGetControllerFromMech (my_mech);
ivar = 130;
CxPmacGetIvar (Mech_cntrl, ivar, &ivalue);
```

```
SEE ALSO

CxPmacSetIvar
**CxPmacSetIvar**

Sets the value of the given PMAC i-variable

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxPmacSetIvar ( CxController cntrl, long ivar, double value)
```

**ARGUMENTS**

- `cntrl` Controller ID
- `ivar` i-variable number
- `value` Value to set to

**DESCRIPTION**

This function is used to set the value of a given i-variable (initialization variable). A valid controller ID pointer should be obtained by calling `CxGetControllerFromMech` (see the Controller Management section of the CODE API Programmer’s Reference Manual – Volume I for more details) before this function is called. Please refer to PMAC User’s Manual and PMAC Software Reference for definitions of i-variables and their respective units.

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the `CxGetErrorNumber` function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to the CIMServer.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from the CIMServer.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

This example shows how to set a PMAC i-variable. In this example, I130 is set to 10000.

```c

//
// CxController Mech_cntrl;
long ivar;
double ivalue;
CxMechanism my_mech;

/* my_mech is a mechanism returned by open_mechanism call */
Mech_cntrl = CxGetControllerFromMech ( my_mech );

ivar = 130;
ivalue = 10000.0;
```
CxPmacSetIvar ( Mech_cntrl, ivar, ivalue );
.
.
SEE ALSO
CxGetControllerFromMech, CxPmacGetIvar
MEI DSP-Specific Functions
**Synopsis**

```c
#include <code/dsp_client.h>
long CxDspDisableCamming (CxMechanism mech, int cam_axis)
```

**Arguments**
- `mech` : Mechanism ID
- `cam_axis` : Axis number on the DSP card corresponding to the cam axis

**Description**

This function is used to disable electronic camming on an axis. This axis must not necessarily be part of the mechanism in question; however, it must reside on the same DSP card as the mechanism axes.

Under electronic camming, the commanded trajectories of the cam axis are executed based on the rate of position change on a “master” axis. That is, the magnitude of the master axis velocity is used as the input in generating the cam axis motions. (For further details, please consult the *DSP-Series Motion Controller C Programming Manual* from Motion Engineering, Inc.)

**Return Values**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**Example**

The following code segment demonstrates how to enable and disable electronic camming for a mechanism axis. In this example, the trajectories on axis 1 are computed based on the commanded velocities of axis 0.

```c
/* mech is a mechanism ID returned by CxOpenMechanism() */
CxMechanism mech;
int master = 0;
int cam = 1;
DspCammingSource mode = CAM_COMMAND_POS;
CxDspEnableCamming(mech, master, cam, mode);
/* Send motion commands here */
CxDspDisableCamming(mech, cam);
```

**See Also**

CxDspEnableCamming, CxDspQueuedEnableCamming, CxDspQueuedDisableCamming, CxDspEnableGearing, CxDspDisableGearing, CxDspQueuedEnableGearing, CxDspQueuedDisableGearing
CxDspDisableGearing

Disables electronic gearing.

SYNOPSIS

```c
#include <code/dsp_client.h>
long CxDspDisableGearing (CxMechanism mech, int slave)
```

ARGUMENTS

- **mech**  Mechanism ID
- **slave**  Axis number on the DSP card corresponding to the “slave” axis

DESCRIPTION

This function is used to disable electronic gearing on an axis. This axis must not necessarily be part of the mechanism in question; however, it must reside on the same DSP card as the mechanism axes.

Under electronic gearing, the DSP card moves the slave axis by the change in position of the master axis (either commanded or actual), multiplied by a gearing ratio. The displacement produced by gearing is added to any “normal” trajectory commands. (For further details, please consult the *DSP-Series Motion Controller C Programming Manual* from Motion Engineering, Inc.)

RETURN VALUES

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

EXAMPLE

The following code segment demonstrates how to enable and disable electronic gearing for a mechanism axis. In this example, the positions on axis 2 are computed based on position changes on axis 0.

```c
/* mech is a mechanism ID returned by CxOpenMechanism() */
CxMechanism mech;
int master = 0;
int slave = 2;
double ratio = 2.0;
DspGearingSource mode = GEAR_ACTUAL_POS;

CxDspEnableGearing(mech, master, slave, ratio, mode);
/* Send motion commands here */
CxDspDisableGearing(mech, slave);
```

SEE ALSO

CxDspEnableGearing, CxDspQueuedEnableGearing, CxDspQueuedDisableGearing,
CxDspEnableCamming, CxDspDisableCamming, CxDspQueuedEnableCamming,
CxDspQueuedDisableCamming
**CxDspDisablePredictiveLimits**

Disables predictive software limits used with constant velocity moves

**SYNOPSIS**

```c
#include <code/dsp_client.h>
long CxDspDisablePredictiveLimits (CxMechanism mech, int axis_index)
```

**ARGUMENTS**

- `mech` Mechanism ID
- `axis_index` Axis number on the mechanism corresponding to the motor axis for which predictive limits should be disabled.

**DESCRIPTION**

This function disables the predictive software limits set using `CxDspEnablePredictiveLimits()`.

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (`CX_ERROR`) is returned.

**EXAMPLE**

Explain what the example does here.

Show your example here.

**SEE ALSO**

- `CxDspEnablePredictiveLimits`
- `CxDspQueuedEnablePredictiveLimits`
- `CxDspQueuedDisablePredictiveLimits`
CxDspEnableCamming
Enables electronic camming.

SYNOPSIS

```c
#include <code/dsp_client.h>
long CxDspEnableCamming (CxMechanism mech, int master_axis,
    int cam_axis, DspCammingSource mode)
```

ARGUMENTS

- **mech**: Mechanism ID
- **master_axis**: Axis number on the DSP card corresponding to the master axis
- **cam_axis**: Axis number on the DSP card corresponding to the cam axis
- **mode**: Either `CAM_ACTUAL_POS` or `CAM_COMMAND_POS`, depending on whether the camming is performed based on the actual or the commanded position of the master axis

DESCRIPTION

This function is used to enable electronic camming on an axis. This axis must not necessarily be part of the mechanism in question; however, it must reside on the same DSP card as the mechanism axes.

Under electronic camming, the commanded trajectories of the cam axis are executed based on the rate of position change on a “master” axis. That is, the magnitude of the master axis velocity is used as the input in generating the cam axis motions. (For further details, please consult the **DSP-Series Motion Controller C Programming Manual** from Motion Engineering, Inc.)

WARNING: Under electronic gearing and electronic camming, the on-board software limits are not observed reliably. If the gearing- or camming-induced motions should cause the slave to move beyond the software limits, **this will not be detected by the DSP board**. This shortcoming is due to limitations in the MEI DSP firmware.

If fewer than eight physical axes are in use on the card, it may be possible to enforce software limits using “phantom axes” (described in the Cimetrix **MEI DSP Device Interface Release Notes**). This would require that Cimetrix create a customized version of CIMControl for the customer’s application. The feasibility of this option will depend on the application and the set of features used.

RETURN VALUES

This function returns 0 if successful; otherwise, -1 (**CX_ERROR**) is returned.

EXAMPLE

The following code segment demonstrates how to enable and disable electronic camming for a mechanism axis. In this example, the trajectories on axis 1 are computed based on the commanded velocities of axis 0.

```c
/* mech is a mechanism ID returned by CxOpenMechanism() */
CxMechanism mech;
int master = 0;
int cam = 1;
DspCammingSource mode = CAM_COMMAND_POS;
```
CxDspEnableCamming(mech, master, cam, mode);
/* Send motion commands here */
CxDspDisableCamming(mech, cam);

SEE ALSO

CxDspDisableCamming, CxDspQueuedEnableCamming, CxDspQueuedDisableCamming,
CxDspEnableGearing, CxDspDisableGearing, CxDspQueuedEnableGearing, CxDspQueuedDisableGearing
CxDspEnableGearing

Enables electronic gearing.

SYNOPSIS

#include <code/dsp_client.h>

long CxDspEnableGearing (CxMechanism mech, int master, int slave, double ratio, DspGearingSource mode)

ARGUMENTS

mech  Mechanism ID
master Axis number on the DSP card corresponding to the master axis
slave Axis number on the DSP card corresponding to the slave axis
ratio The ratio of the slave axis displacements to the master axis displacements
mode  Either CAM_ACTUAL_POS or CAM_COMMAND_POS, depending on whether the camming is performed based on the actual or the commanded position of the master axis

DESCRIPTION

This function is used to enable electronic gearing on an axis. This axis must not necessarily be part of the mechanism in question; however, it must reside on the same DSP card as the mechanism axes.

Under electronic gearing, the DSP card moves the slave axis by the change in position of the master axis (either commanded or actual), multiplied by a gearing ratio. (For further details, please consult the DSP-Series Motion Controller C Programming Manual from Motion Engineering, Inc.)

WARNING: Under electronic gearing and electronic camming, the on-board software limits are not observed reliably. If the gearing- or camming-induced motions should cause the slave to move beyond the software limits, *this will not be detected by the DSP board*. This shortcoming is due to limitations in the MEI DSP firmware.

If fewer than eight physical axes are in use on the card, it may be possible to enforce software limits using “phantom axes” (described in the Cimetrix MEI DSP Device Interface Release Notes). This would require that Cimetrix create a customized version of CIMControl for the customer’s application. The feasibility of this option will depend on the application and the set of features used.

RETURN VALUES

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

EXAMPLE

The following code segment demonstrates how to enable and disable electronic gearing for a mechanism axis. In this example, the positions on axis 2 are computed based on position changes on axis 0.

    /* mech is a mechanism ID returned by CxOpenMechanism() */
    CxMechanism mech;
    int master = 0;
    int slave = 2;
    double ratio = 2.0;
    DspGearingSource mode = GEAR_ACTUAL_POS;
CxDspEnableGearing(mech, master, slave, ratio, mode);
/* Send motion commands here */
CxDspDisableGearing(mech, slave);

SEE ALSO
CxDspDisableGearing, CxDspQueuedEnableGearing, CxDspQueuedDisableGearing, CxDspEnableCamming,
CxDspDisableCamming, CxDspQueuedEnableCamming, CxDspQueuedDisableCamming
CxDspEnablePredictiveLimits
Enables predictive software limits on an axis

SYNOPSIS
#include <code/dsp_client.h>
long CxDspEnablePredictiveLimits (CxMechanism mech, int axis_index,
        double margin)

ARGUMENTS
    mech          Mechanism ID
    axis_index    Axis number on the mechanism corresponding to the motor axis for which
                  predictive limits should be disabled.
    margin        Error margin in position sensor units

DESCRIPTION
The software limits on the MEI DSP card can only cause an axis to decelerate once the limit has been
reached; hence, when the axes come to a halt, they would actually lie beyond the travel limit range. This
can be a problem when using high-speed constant velocity moves, since this can cause the axis to lie well
beyond the permissible travel range.

To circumvent this limitation, this API function allows the user to enabling “predictive software limits” for
constant velocity moves. This function works by adjusting the on-board software limits based on the move
velocity and the programmed stop rate. The margin parameter specifies a margin for error, and is expressed in
position sensor units (which are typically encoder counts). It is a useful means of compensating for processor
latency, since the adjusted software limits can not be computed instantaneously.

RETURN VALUES
This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

EXAMPLE
Explain what the example does here.

Show your example here.

SEE ALSO
CxDspDisablePredictiveLimits, CxDspQueuedEnablePredictiveLimits,
CxDspQueuedDisablePredictiveLimits
CxDspGetEStopRate
Enter definition here

SYNOPSIS
Enter Syntax here, for example:
#include <code/robpac.h>
long CxCloseMechanism (CxMechanism mech_to_close)

ARGUMENTS
Enter argument name here.
Enter argument definition here.

DESCRIPTION
Enter description of API here.

RETURN VALUES
Enter return values here, for example:
This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS
If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter error code name here.</td>
<td>Enter error code description here.</td>
</tr>
</tbody>
</table>

EXAMPLE
Explain what the example does here.
Show your example here.

SEE ALSO
List other APIs to reference here.
CXDSPGETNEGATIVESWLIMIT

Enter definition here

SYNOPSIS

Enter Syntax here, for example:
#include <code/robpac.h>
long CxCloseMechanism (CxMechanism mech_to_close)

ARGUMENTS

Enter argument name here.
Enter argument definition here.

DESCRIPTION

Enter description of API here.

RETURN VALUES

Enter return values here, for example:
This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter error code name here.</td>
<td>Enter error code description here.</td>
</tr>
</tbody>
</table>

EXAMPLE

Explain what the example does here.

Show your example here.

SEE ALSO

List other APIs to reference here.
**CxDspGetNegativeSWLimitPolicy**

Enter definition here

**SYNOPSIS**
Enter Syntax here, for example:
```c
#include <code/robpac.h>
long CxCloseMechanism ( CXMechanism mech_to_close )
```

**ARGUMENTS**
Enter argument name here.

**DESCRIPTION**
Enter description of API here.

**RETURN VALUES**
Enter return values here, for example:
This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**
If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter error code name here</td>
<td>Enter error code description here</td>
</tr>
</tbody>
</table>

**EXAMPLE**
Explain what the example does here.

Show your example here.

**SEE ALSO**
List other APIs to reference here.
**CxDspGetPositiveSWLimit**

Enter definition here

**SYNOPSIS**

Enter Syntax here, for example:

```
#include <code/robpac.h>
long CxCloseMechanism (CxMechanism mech_to_close)
```

**ARGUMENTS**

Enter argument name here. Enter argument definition here.

**DESCRIPTION**

Enter description of API here.

**RETURN VALUES**

Enter return values here, for example:

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter error code name here.</td>
<td>Enter error code description here.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**EXAMPLE**

Explain what the example does here.

Show your example here.

**SEE ALSO**

List other APIs to reference here.
**CXDspGetPositiveSWLimitPolicy**
Enter definition here

**SYNOPSIS**
Enter Syntax here, for example:
#include <code/robpac.h>
long CxCloseMechanism (CxMechanism mech_to_close)

**ARGUMENTS**
Enter argument name here.
 Enter argument definition here.

**DESCRIPTION**
Enter description of API here.

**RETURN VALUES**
Enter return values here, for example:
This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**
If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter error code name here.</td>
<td>Enter error code description here.</td>
</tr>
</tbody>
</table>

**EXAMPLE**
Explain what the example does here.
Show your example here.

**SEE ALSO**
List other APIs to reference here.
**Synopsis**
Enter Syntax here, for example:
#include <code/robpac.h>
long CxCloseMechanism (CxMechanism mech_to_close)

**Arguments**
Enter argument name here.
Enter argument definition here.

**Description**
Enter description of API here.

**Return Values**
Enter return values here, for example:
This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**Errors**
If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter error code name here</td>
<td>Enter error code description here</td>
</tr>
</tbody>
</table>

**Example**
Explain what the example does here.
Show your example here.

**See Also**
List other APIs to reference here.
**CxDspGetStopRate**

Enter definition here

**SYNOPSIS**

Enter Syntax here, for example:

```c
#include <code/robpac.h>
long CxCloseMechanism (CxMechanism mech_to_close)
```

**ARGUMENTS**

Enter argument name here.

Enter argument definition here.

**DESCRIPTION**

Enter description of API here.

**RETURN VALUES**

Enter return values here, for example:

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter error code name here.</td>
<td>Enter error code description here.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

Explain what the example does here.

Show your example here.

**SEE ALSO**

List other APIs to reference here.
**CxDspQueuedDisableCamming**

Enter definition here

**SYNOPSIS**
Enter Syntax here, for example:
```
#include <code/robpac.h>

long CxCloseMechanism (CxMechanism mech_to_close)
```

**ARGUMENTS**
Enter argument name here.
Enter argument definition here.

**DESCRIPTION**
Enter description of API here.

**RETURN VALUES**
Enter return values here, for example:
This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**
If the function returns an error condition, the error code can be obtained by using the CxGetErrorCode function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter error code name here.</td>
<td>Enter error code description here.</td>
</tr>
</tbody>
</table>

**EXAMPLE**
Explain what the example does here.

Show your example here.

**SEE ALSO**
List other APIs to reference here.
**CxDspQueuedDisableGearing**

Enter definition here

**SYNOPSIS**
Enter Syntax here, for example:
```c
#include <code/robpac.h>
long CxCloseMechanism (CxMechanism mech_to_close)
```

**ARGUMENTS**
Enter argument name here.
Enter argument definition here.

**DESCRIPTION**
Enter description of API here.

**RETURN VALUES**
Enter return values here, for example:
This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**
If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter error code name here.</td>
<td>Enter error code description here.</td>
</tr>
</tbody>
</table>

**EXAMPLE**
Explain what the example does here.

Show your example here.

**SEE ALSO**
List other APIs to reference here.
CXDSPQUEUEDDISABLEPREDICTIVELIMITS

SYNOPSIS
Enter Syntax here, for example:
#include <code/robpac.h>
long CxCloseMechanism (CxMechanism mech_to_close)

ARGUMENTS
Enter argument name here.
Enter argument definition here.

DESCRIPTION
Enter description of API here.

RETURN VALUES
Enter return values here, for example:
This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS
If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter error code name here.</td>
<td>Enter error code description here.</td>
</tr>
</tbody>
</table>

EXAMPLE
Explain what the example does here.
Show your example here.

SEE ALSO
List other APIs to reference here.
CxDspQueuedEnableCamming

Enter definition here

SYNOPSIS
Enter Syntax here, for example:
#include <code/robpac.h>
long CxCloseMechanism (CxMechanism mech_to_close)

ARGUMENTS
Enter argument name here.
Enter argument definition here.

DESCRIPTION
Enter description of API here.

RETURN VALUES
Enter return values here, for example:
This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS
If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter error code name here.</td>
<td>Enter error code description here.</td>
</tr>
</tbody>
</table>

EXAMPLE
Explain what the example does here.

Show your example here.

SEE ALSO
List other APIs to reference here.
CxDspQueuedEnableGearing

Enter definition here

SYNOPSIS
Enter Syntax here, for example:
#include <code/robpac.h>
long CxCloseMechanism (CxMechanism mech_to_close)

ARGUMENTS
Enter argument name here.
Enter argument definition here.

DESCRIPTION
Enter description of API here.

RETURN VALUES
Enter return values here, for example:
This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS
If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter error code name here.</td>
<td>Enter error code description here.</td>
</tr>
</tbody>
</table>

EXAMPLE
Explain what the example does here.

Show your example here.

SEE ALSO
List other APIs to reference here.
**CxQueuedEnablePredictiveLimits**

- **SYNOPSIS**
  Enter Syntax here, for example:
  ```c
  #include <code/robpac.h>
  long CxCloseMechanism (CxMechanism mech_to_close)
  ```

- **ARGUMENTS**
  Enter argument name here.
  Enter argument definition here.

- **DESCRIPTION**
  Enter description of API here.

- **RETURN VALUES**
  Enter return values here, for example:
  This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

- **ERRORS**
  If the function returns an error condition, the error code can be obtained by using the `CxGetErrorCode` function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter error code name here.</td>
<td>Enter error code description here.</td>
</tr>
</tbody>
</table>

- **EXAMPLE**
  Explain what the example does here.

  Show your example here.

- **SEE ALSO**
  List other APIs to reference here.
**CxDspQueuedSetNegativeSWLimitPolicy**

Enter definition here

**SYNOPSIS**
Enter Syntax here, for example:
```c
#include <code/robpac.h>
long CxCloseMechanism (CxMechanism mech_to_close)
```

**ARGUMENTS**
Enter argument name here.

**DESCRIPTION**
Enter description of API here.

**RETURN VALUES**
Enter return values here, for example:
This function returns **0** if successful; otherwise, **-1 (CX_ERROR)** is returned.

**ERRORS**
If the function returns an error condition, the error code can be obtained by using the `CxGetErrorNumber` function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter error code name here.</td>
<td>Enter error code description here.</td>
</tr>
</tbody>
</table>

**EXAMPLE**
Explain what the example does here.

Show your example here.

**SEE ALSO**
List other APIs to reference here.
**CxDspQueuedSetPositiveSWLimitPolicy**

Enter definition here

**SYNOPSIS**

Enter Syntax here, for example:
```
#include <code/robpac.h>
long CxCloseMechanism (CxMechanism mech_to_close)
```

**ARGUMENTS**

Enter argument name here. Enter argument definition here.

**DESCRIPTION**

Enter description of API here.

**RETURN VALUES**

Enter return values here, for example:
This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorCode function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter error code name here.</td>
<td>Enter error code description here.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

Explain what the example does here.
Show your example here.

**SEE ALSO**

List other APIs to reference here.
**CxDspQueuedSetShortMoveMode**

Enter definition here

**SYNOPSIS**

Enter Syntax here, for example:

```
#include <code/robpac.h>
long CxCloseMechanism (CxMechanism mech_to_close)
```

**ARGUMENTS**

Enter argument name here.

Enter argument definition here.

**DESCRIPTION**

Enter description of API here.

**RETURN VALUES**

Enter return values here, for example:

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter error code name here.</td>
<td>Enter error code description here.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

Explain what the example does here.

Show your example here.

**SEE ALSO**

List other APIs to reference here.
**Synopsis**

Enter Syntax here, for example:
```c
#include <code/robpac.h>
long CxCloseMechanism (CxMechanism mech_to_close)
```

**Arguments**

Enter argument name here.

Enter argument definition here.

**Description**

Enter description of API here.

**Return Values**

Enter return values here, for example:

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**Errors**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter error code name here.</td>
<td>Enter error code description here.</td>
</tr>
</tbody>
</table>

**Example**

Explain what the example does here.

Show your example here.

**See Also**

List other APIs to reference here.
**CxDspSetNegativeSWLimit**

Enter definition here

**SYNOPSIS**

Enter Syntax here, for example:
#include <code/robpac.h>
long CxCloseMechanism (CxMechanism mech_to_close)

**ARGUMENTS**

Enter argument name here.

Enter argument definition here.

**DESCRIPTION**

Enter description of API here.

**RETURN VALUES**

Enter return values here, for example:
This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter error code name here.</td>
<td>Enter error code description here.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

Explain what the example does here.

Show your example here.

**SEE ALSO**

List other APIs to reference here.
**Synopsis**

Enter Syntax here, for example:
```
#include <code/robpac.h>
long CxCloseMechanism (CxMechanism mech_to_close)
```

**Arguments**

Enter argument name here.

Enter argument definition here.

**Description**

Enter description of API here.

**Return Values**

Enter return values here, for example:
This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**Errors**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter error code name here.</td>
<td>Enter error code description here.</td>
</tr>
</tbody>
</table>

**Example**

Explain what the example does here.

Show your example here.

**See Also**

List other APIs to reference here.
CxDspSetPositiveSWLimit
Enter definition here

SYNOPSIS
Enter Syntax here, for example:
#include <code/robpac.h>
long CxCloseMechanism (CxMechanism mech_to_close)

ARGUMENTS
Enter argument name here.
Enter argument definition here.

DESCRIPTION
Enter description of API here.

RETURN VALUES
Enter return values here, for example:
This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS
If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter error code name here.</td>
<td>Enter error code description here.</td>
</tr>
</tbody>
</table>

EXAMPLE
Explain what the example does here.
Show your example here.

SEE ALSO
List other APIs to reference here.
**CxDspSetPositiveSWLimitPolicy**

Enter definition here

**SYNOPSIS**

Enter Syntax here, for example:

```
#include <code/robpac.h>
long CxCloseMechanism (CxMechanism mech_to_close)
```

**ARGUMENTS**

Enter argument name here.

Enter argument definition here.

**DESCRIPTION**

Enter description of API here.

**RETURN VALUES**

Enter return values here, for example:

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter error code name here.</td>
<td>Enter error code description here.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

Explain what the example does here.

Show your example here.

**SEE ALSO**

List other APIs to reference here.
CxDspSetShortMoveMode
Enter definition here

SYNOPSIS
Enter Syntax here, for example:
#include <code/robpac.h>
long CxCloseMechanism (CxMechanism mech_to_close)

ARGUMENTS
Enter argument name here.
Enter argument definition here.

DESCRIPTION
Enter description of API here.

RETURN VALUES
Enter return values here, for example:
This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS
If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter error code name here.</td>
<td>Enter error code description here.</td>
</tr>
</tbody>
</table>

EXAMPLE
Explain what the example does here.

Show your example here.

SEE ALSO
List other APIs to reference here.
**CXDspSetStopRate**

Enter definition here

**SYNOPSIS**

Enter Syntax here, for example:
#include <code/robpac.h>
long CxCloseMechanism (CxMechanism mech_to_close)

**ARGUMENTS**

Enter argument name here. Enter argument definition here.

**DESCRIPTION**

Enter description of API here.

**RETURN VALUES**

Enter return values here, for example:
This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter error code name here.</td>
<td>Enter error code description here.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

Explain what the example does here.

Show your example here.

**SEE ALSO**

List other APIs to reference here.
**CxGetDacLimit**

Enter definition here

**SYNOPSIS**

Enter Syntax here, for example:

```c
#include <code/robpac.h>
long CxCloseMechanism (CxMechanism mech_to_close)
```

**ARGUMENTS**

Enter argument name here.

Enter argument definition here.

**DESCRIPTION**

Enter description of API here.

**RETURN VALUES**

Enter return values here, for example:

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter error code name here.</td>
<td>Enter error code description here.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

Explain what the example does here.

Show your example here.

**SEE ALSO**

List other APIs to reference here.
**CxSetDacLimit**

Enter definition here

**SYNOPSIS**

Enter Syntax here, for example:
```
#include <code/robpac.h>
long CxCloseMechanism (CxMechanism mech_to_close)
```

**ARGUMENTS**

Enter argument name here.

**DESCRIPTION**

Enter description of API here.

**RETURN VALUES**

Enter return values here, for example:

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter error code name here.</td>
<td>Enter error code description here.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

Explain what the example does here.

Show your example here.

**SEE ALSO**

List other APIs to reference here.
MEI XMP-Specific Functions
**CxMpiGetAxisConfig**

Returns the XMP axis configuration for a mechanism joint

**SYNOPSIS**

```c
#include <code/xmp.h>
long CxMpiGetAxisConfig (CxMechanism mech, long joint,
                        struct MPIAxisInPosition *config, long configSize)
```

**ARGUMENTS**

- **mech** Mechanism ID
- **joint** Joint for which to retrieve configuration information
- **config** Pointer to structure that will be filled with the axis configuration

**DESCRIPTION**

This function is executed immediately and fills the `config` structure with the MEI axis configuration for the specified joint. The mechanism may be opened for monitor or control. In CIMulation, the configuration structure will be filled with zeros.

Data structure type MPIAxisInPosition is defined in stdmei.h, which comes with the standard MEI Xmp release.

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>Invalid parameter was passed to this function.</td>
</tr>
<tr>
<td>CX_INVALID_JOINT_NUMBER</td>
<td>Specified joint number is not valid.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following segment gets the axis configuration for joint 0:

```c
long i;
CxMechanism mech;
MPIAxisInPosition config;
.
.
CxMpiGetAxisConfig (mech, 0, &config, sizeof (config));
printf ("Joint 0 positionFine: %lf\n", config.tolerance.positionFine);
```
printf ("Joint 0 positionCoarse: %ld\n", config.tolerance.positionCoarse);
printf ("Joint 0 settlingTime: %lf\n", config.settlingTime);

SEE ALSO
CxMpiSetAxisConfig, CxMpiQueuedSetAxisConfig
CxMpiQueuedSetAxisConfig
Sets the XMP axis configuration for a mechanism joint

SYNOPSIS
#include <code/xmp.h>
long CxMpiQueuedSetAxisConfig (CxMechanism mech, long joint, long mask, 
MPIAxisInPosition *config, long configSize)

ARGUMENTS
mech Mechanism ID
joint Joint for which to retrieve configuration information
mask Mask which contains information about what fields need to be set
config Pointer to structure that will be used to set the axis configuration
configSize Size of the structure pointed to by config

DESCRIPTION
This function is executed through the mechanism motion queue and uses the mask parameter and config structure to set the MEI axis configuration for the specified joint. The mechanism must be opened for control. In CIMulation, this function has no effect.

Like CxMpiSetAxisConfig, CxMpiQueuedSetAxisConfig will set the “fine position”, “coarse position” and “settling time” attributes of a joint. The valid mask for this API: XmpPosFine, XmpPosCoarse and XmpSettlingTime.

Data structure type MPIAxisInPosition is defined in stdmei.h, which comes with the standard MEI Xmp release.

RETURN VALUES
This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS
If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server.</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>Mechanism is not opened for control.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>Invalid parameter was passed to this function.</td>
</tr>
<tr>
<td>CX_INVALID_JOINT_NUMBER</td>
<td>Specified joint number is not valid.</td>
</tr>
</tbody>
</table>

EXAMPLE
This code segment changes the coarse position for joint 0 between two queued moves.
. 
CxServer server;
CxMechanism mech;
double target1[CX_MAX_JOINTS];
double target2[CX_MAX_JOINTS];
MPIAxisInPosition config;
long mask;
.
/* mech has been opened for CX_CONTROL prior to this code segment */

CxSetBlendPolicy (mech, CX_MOVE_TO);

cxMpiGetAxisConfig (mech, 0, &config, sizeof (config));

cxconfig.tolerance.positionCoarse = 200;

mask = XmpPosCoarse;

cxMoveAllAxes (mech, target1);

cxMpiQueuedSetAxisConfig (mech, 0, mask, &config, sizeof(config));
cxMoveAllAxes (mech, target2);

CxWaitForEndOfMotion (mech);

SEE ALSO

CxMpiGetAxisConfig, CxMpiSetAxisConfig
**CxMpiGetAxisConfig**

Sets the XMP axis configuration for a mechanism joint

**SYNOPSIS**

```c
#include <code/xmp.h>
long CxMpiGetAxisConfig (CxMechanism mech, long joint, long mask, 
                        struct MPIAxisInPosition *config, long configSize);
```

**ARGUMENTS**

- **mech**
  - Mechanism ID
- **joint**
  - Joint for which to retrieve the configuration information
- **mask**
  - Mask which contains information about which fields need to be set
- **config**
  - Pointer to structure that will be used to set the axis configuration
- **configSize**
  - Size of the structure pointed to by **config**

**DESCRIPTION**

This function is executed immediately and uses the **mask** parameter and **config** structure to set the MEI axis configuration for the specified joint. The mechanism must be opened for control. In CIMulation, this function has no effect.

Unlike CxXmpSetAxisConfig, CxMpiSetAxisConfig will set the "fine position", "coarse position" and "settling time" attributes of a joint. The valid mask for this API: XmpPosFine, XmpPosCoarse and XmpSettlingTime.

Data structure type MPIAxisInPosition is defined in stdmei.h, which comes with the standard MEI Xmp release.

**RETURN VALUES**

This function returns **0** if successful; otherwise, **-1 (CX_ERROR)** is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server.</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>Mechanism is not opened for control.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>Invalid parameter was passed to this function.</td>
</tr>
<tr>
<td>CX_INVALID_JOINT_NUMBER</td>
<td>Specified joint number is not valid.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

This code segment changes the coarse position, fine position and settling time for joint 0.
/* mech has been opened for CX_CONTROL prior to this code segment */
CxMpiGetAxisConfig (mech, 0, &config, sizeof (config));

config.tolerance.positionFine = 200;
config.tolerance.positionCoarse = 200;
config.settlingTime = 0;

mask = 0;
mask = | XmpPosFine;
mask = | XmpPosCoarse;
mask = | XmpSettlingTime;
CxMpiSetAxisConfig (mech, 0, mask, &config, sizeof(config));

SEE ALSO

CxMpiGetAxisConfig, CxMpiQueuedSetAxisConfig
**CXMPGetAxisConfig**

Returns the XMP axis configuration for a mechanism joint

**SYNOPSIS**

```c
#include <code/xmp.h>
long CXMPGetAxisConfig (CxMechanism mech, long joint, MEIPositionConfig *config, long configSize)
```

**ARGUMENTS**

- `mech`: Mechanism ID
- `joint`: Joint for which to retrieve configuration information
- `config`: Pointer to structure that will be filled with the axis configuration
- `configSize`: Size of the structure pointed to by `config`

**DESCRIPTION**

This function is executed immediately and fills the `config` structure with the MEI axis configuration for the specified joint. The mechanism may be opened for monitor or control. In CIMulation, the configuration structure will be filled with zeros. In CIMControl, the Axis, APos, Commutation, and Interpolation structure members within `config` are filled with zeros.

**RETURN VALUES**

This function returns `0` if successful; otherwise, `-1 (CX_ERROR)` is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the `CxGetErrorNumber` function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>Invalid parameter was passed to this function.</td>
</tr>
<tr>
<td>CX_INVALID_JOINT_NUMBER</td>
<td>Specified joint number is not valid.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following segment gets the axis configuration for joint 0:

```c
long i;
CxMechanism mech;
MEIPositionConfig config;
.
.
CXMPGetAxisConfig (mech, 0, &config, sizeof (config));
```
printf ("Joint 0 GainSwitchType: %d\n", config.GainSwitchType);
printf ("Joint 0 GainIndex: %d\n", config.GainIndex);

SEE ALSO

CxXmpSetAxisConfig, CxXmpQueuedSetAxisConfig
CxXmpGetJerkPercent

Returns the jerk percent of the mechanism

SYNOPSIS

#include <code/xmp.h>
long CxXmpGetJerkPercent (CxMechanism mech, double *jerk_percent)

ARGUMENTS

mech Mechanism ID
jerk_percent Pointer to double variable that will be filled with the mechanism's jerk percent

DESCRIPTION

This function is executed immediately and fills the jerk_percent variable with the jerk percent which will be used by subsequent coordinated moves. The mechanism may be opened for monitor or control. In CIMulation, the jerk percent will always be set to 0.

This function allows you to set the percent of time spent in the S curve. For example:

```
  25       50       25
```

The example shows a standard Scurve acceleration ramp with the jerk percent set to 50. 25% is in front of the S curve, 25% is in back, and the s-curve makes up 50% of the time.

RETURN VALUES

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>Invalid parameter was passed to this function.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following segment gets the jerk percent:

```c
double jerk_percent;
CxMechanism mech;
.
.
CxXmpGetJerkPercent (mech, &jerk_percent);
```
printf ("Jerk percentage is %f.\n", jerk_percent);

SEE ALSO

CxXmpSetJerkPercent
**CxXmpGetJointInterpType**

Returns the XMP joint interpolation type for the mechanism

SYNOPSIS

```c
#include <code/xmp.h>
long CxXmpGetJointInterpType (CxMechanism mech, long *joint_interp_type)
```

ARGUMENTS

- `mech`: Mechanism ID
- `joint_interp_type`: Pointer to long variable that will be filled with the mechanism's joint interpolation type

DESCRIPTION

This function is executed immediately and fills the `joint_interp_type` variable with the joint interpolation type which will be used by subsequent moves. The mechanism may be opened for monitoring or control. In CIMulation, the joint interpolation type will always be set to 0.

RETURN VALUES

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS

If the function returns an error condition, the error code can be obtained by using the `CxGetErrorNumber` function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>Invalid parameter was passed to this function.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following segment gets the joint interpolation type:

```c
long joint_interp_type;
CxMechanism mech;
.
.
CxXmpGetJointInterpType (mech, &joint_interp_type);
printf ("Joint moves are %s\n", (joint_interp_type == XmpJointMoveTypeCOORDINATED) ? "coordinated" : "rapid");
```

SEE ALSO

`CxXmpSetJointInterpType`
**CxXmpGetRapidJerkPercent**

Returns the jerk percent used by the joint for rapid moves

**SYNOPSIS**

```c
#include <code/xmp.h>
long CxXmpGetJerkPercent (CxMechanism mech, long joint, double *jerk_percent)
```

**ARGUMENTS**

- `mech` : Mechanism ID
- `joint` : Joint to which the jerk percent applies
- `jerk_percent` : Pointer to double variable that will be filled with the joint’s jerk percent

**DESCRIPTION**

This function is executed immediately, and fills the `jerk_percent` variable with the jerk percent which will be used by the joint for subsequent rapid moves. The mechanism may be opened for monitor or control. In CIMulation, the jerk percent will always be set to 0.

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>Invalid parameter was passed to this function.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following segment gets the jerk percent:

```c
double jerk_percent;
CxMechanism mech;
.
.
CxXmpGetRapidJerkPercent (mech, 0, &jerk_percent);
printf("Jerk percentage of joint 0 is \%.2f\n", jerk_percent);
```

**SEE ALSO**

CxXmpSetRapidJerkPercent
**CxXmpQueuedSetAxisConfig**

Sets the XMP axis configuration for a mechanism joint

**SYNOPSIS**

```c
#include <code/xmp.h>
long CxXmpQueuedSetAxisConfig (CxMechanism mech, long joint, long mask, MEIPositionConfig *config, long configSize)
```

**ARGUMENTS**

- **mech**: Mechanism ID
- **joint**: Joint for which to retrieve configuration information
- **mask**: Mask which contains information about what fields need to be set
- **config**: Pointer to structure that will be used to set the axis configuration
- **configSize**: Size of the structure pointed to by `config`

**DESCRIPTION**

This function is executed through the mechanism motion queue and uses the `mask` parameter and `config` structure to set the MEI axis configuration for the specified joint. The mechanism must be opened for control. In CImulation, this function has no effect.

**RETURN VALUES**

This function returns **0** if successful; otherwise, `-1 (CX_ERROR)` is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the `CxGetErrorNumber` function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server.</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>Mechanism is not opened for control.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>Invalid parameter was passed to this function.</td>
</tr>
<tr>
<td>CX_INVALID_JOINT_NUMBER</td>
<td>Specified joint number is not valid.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

This code segment changes the Kp MOVING gain for joint 0 between two queued moves.

```c
CxServer server;
CxMechanism mech;
double target1[CX_MAX_JOINTS];
```
double target2[CX_MAX_JOINTS];
MEIPositionConfig config;
long mask;

/* mech has been opened for CX_CONTROL prior to this code segment */

CxSetBlendPolicy (mech, CX_MOVE_TO);

CxXmpGetAxisConfig (mech, 0, &config, sizeof (config));
config.Coeff[MEIXmpGainMOVING].PID.Kp = 64;

mask = XmpCoeff;

CxMoveAllAxes (mech, target1);
CxXmpQueuedSetAxisConfig (mech, 0, mask, &config, sizeof(config));
CxMoveAllAxes (mech, target2);

CxWaitForEndOfMotion (mech);

SEE ALSO

CxXmpGetAxisConfig, CxXmpSetAxisConfig
**CxXmpSetAxisConfig**

Sets the XMP axis configuration for a mechanism joint

**SYNOPSIS**

```
#include <code/xmp.h>
long CxXmpSetAxisConfig (CxMechanism mech, long joint, long mask,
        MEIPositionConfig *config, long configSize)
```

**ARGUMENTS**

- `mech` : Mechanism ID
- `joint` : Joint for which to retrieve configuration information
- `mask` : Mask which contains information about what fields need to be set
- `config` : Pointer to structure that will be used to set the axis configuration
- `configSize` : Size of the structure pointed to by `config`

**DESCRIPTION**

This function is executed immediately and uses the `mask` parameter and `config` structure to set the MEI axis configuration for the specified joint. The mechanism must be opened for control. In CIMulation, this function has no effect.

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the `CxGetErrorNumber` function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server.</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>Mechanism is not opened for control.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>Invalid parameter was passed to this function.</td>
</tr>
<tr>
<td>CX_INVALID.Join.Number</td>
<td>Specified joint number is not valid.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

This code segment configures joint 0 to use gain switching and changes the Kp parameter in the MOVING gains.

```c
CxServer server;
CxMechanism mech;
MEIPositionConfig config;
```
long mask;

/* mech has been opened for CX_CONTROL prior to this code segment */
CxXmpGetAxisConfig (mech, 0, &config, sizeof (config));

config.GainSwitchType = MEIXmpSwitchTypeMOTION_ONLY;
config.Coeff[MEIXmpGainMOVING].PID.Kp = 64;

mask = XmpGainIndex | XmpCoeff;
CxXmpSetAxisConfig (mech, 0, mask, &config, sizeof(config));

SEE ALSO
   CxXmpGetAxisConfig, CxXmpQueuedSetAxisConfig
**CxXmpSetJerkPercent**

Sets the mechanism’s jerk percent

**SYNOPSIS**

```c
#include <code/xmp.h>
long CxXmpSetJerkPercent (CxMechanism mech, double jerk_percent)
```

**ARGUMENTS**

- `mech` : Mechanism ID
- `jerk_percent` : Jerk percent to be set for the mechanism

**DESCRIPTION**

This function sets the jerk percent for subsequent coordinated end point moves with the `CX_CONST_RAMP_ACCEL` acceleration type. The valid values for `jerk_percent` are between 0.0 and 100.0 (inclusive). The jerk percent is the amount of acceleration/deceleration phase which will be spent in non-constant acceleration. A jerk percent of 0 will result in trapezoidal moves. A jerk percent of 100 will result in a "pure" S-curve. A jerk percent of 50 will result in the first and last 25% of the acceleration phase being constant jerk. The acceleration of the constant acceleration phase is set with the `CxSetJntAccel` API function. This function is executed through the mechanism motion queue. If the motion queue is aborted before the function is executed, the previous jerk percent will remain in effect. The mechanism must be opened for control. In CIMulation, this function has no effect.

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the `CxGetErrorNumber` function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server.</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>Mechanism is not opened for control.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>Invalid parameter was passed to this function.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

This code segment sets the mechanism to use a 50% S-curve.

```c
CxServer server;
CxMechanism mech;
```
/* mech has been opened for CX_CONTROL prior to this code segment */
CxXmpSetJerkPercent (mech, 50.0);

SEE ALSO
    CxXmpGetJerkPercent,CxSetAccelType,CxSetJntAccel
**CXxmpSetJointInterpType**

Sets the mechanism’s XMP joint interpolation type

**SYNOPSIS**

```c
#include <cx/xmp.h>
long CXxmpSetJointInterpType (CxMechanism mech, long joint_interp_type)
```

**ARGUMENTS**

- `mech` Mechanism ID
- `joint_interp_type` Joint interpolation type to be set for the mechanism

**DESCRIPTION**

This function sets the joint interpolation type for subsequent joint moves. The valid values for `joint_interp_type` are `XmpJointInterpTypeCOORDINATED` and `XmpJointInterpTypeRAPID`. With the coordinated interpolation type, all the axes in the mechanism will start and stop at the same time. With the rapid interpolation type, the axes will start together but all axes will travel at their respective settings and will stop at different times. This function is executed through the mechanism motion queue. If the motion queue is aborted before the function is executed, the previous joint interpolation type will remain in effect. The mechanism must be opened for control. In CIMulation, this function has no effect.

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the `CxGetErrorNumber` function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server.</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>Mechanism is not opened for control.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>Invalid parameter was passed to this function.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

This code segment sets the mechanism to use rapid moves.

```c

```
/* mech has been opened for CX_CONTROL prior to this code segment */
CxXmpSetJointInterpType (mech, XmpJointInterpTypeRAPID);

SEE ALSO
    CxXmpGetJointInterpType
**CxXmpSetRapidJerkPercent**

Sets a joint’s jerk percent

**SYNOPSIS**

```c
#include <code/xmp.h>
long CxXmpSetRapidJerkPercent (CxMechanism mech, long joint, double jerk_percent)
```

**ARGUMENTS**

- `mech` : Mechanism ID
- `joint` : Joint to which the jerk percent will apply
- `jerk_percent` : Jerk percent to be set for the joint

**DESCRIPTION**

This function sets the jerk percent for subsequent rapid end point moves with the `CX_CONST_RAMP_ACCEL` acceleration type. The valid values for `jerk_percent` are between 0.0 and 100.0 (inclusive). The jerk percent is the amount of acceleration/deceleration phase which will be spent in non-constant acceleration. A jerk percent of 0 will result in trapezoidal moves. A jerk percent of 100 will result in a “pure” S-curve. A jerk percent of 50 will result in the first and last 25% of the acceleration phase being constant jerk. The acceleration of the constant acceleration phase is set with the `CxSetJntAccel` API function. This function is executed through the mechanism motion queue. If the motion queue is aborted before the function is executed, the previous jerk percent will remain in effect. The mechanism must be opened for control. In CIMulation, this function has no effect.

**RETURN VALUES**

This function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

If the function returns an error condition, the error code can be obtained by using the `CxGetErrorNumber` function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server.</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>Mechanism is not opened for control.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>Invalid parameter was passed to this function.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

This code segment sets the first joint of the mechanism to use a 50% S-curve during rapid moves.

```c
CxServer server;
CxMechanism mech;
```
/* mech has been opened for CX_CONTROL prior to this code segment */
CxXmpSetRapidJerkPercent (mech, 0, 50.0);

SEE ALSO
   CxXmpGetRapidJerkPercent, CxSetAccelType, CxSetJntAccel
Anorad- IPC Specific Functions
**CxAnoradDisableJoint**

Disables the amplifier of a single joint

**SYNOPSIS**

**C Syntax**

```c
#include <code/anorad.h>
long CxAnoradDisableJoint (CxMechanism mech, long joint)
```

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mech</td>
<td>Mechanism ID</td>
</tr>
<tr>
<td>joint</td>
<td>The joint number in the mechanism to disable. Joint numbers start from 0.</td>
</tr>
</tbody>
</table>

**NOTE:** Joint numbers do not necessarily correspond to motor numbers. To determine the joint number, please refer to the initialization or ".oac" file.

**DESCRIPTION**

This function allows the application developer to disable a single amp or joint at a time.

**NOTE:** Motion cannot occur unless all mechanism amplifiers are enabled.

This function is implemented through the CxSendMechanism command specific for the Anorad IPC motion controller.

**RETURN VALUES**

This function returns CX_OK (0) if successful; otherwise, CX_ERROR (-1) is returned.

**ERRORS**

If the function return CX_ERROR (-1), the error code can be obtained by using the CxGetErrNo function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending a message to the CIMServer</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from the CIMServer</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>The input values are invalid arguments in the function definition.</td>
</tr>
</tbody>
</table>

**KNOWN PROBLEMS**

None

**EXAMPLES**

C Syntax

```c
#include <code/robpac.h>
#include <code/anorad.h>

void main( void )
{
    /* Open CxServer and CxMechanism */
    ...
    CxAnoradDisableJoint( Mech, 0 );
```
SEE ALSO

CxAnoradDisableJoint
**CxAnoradDisableDataCollection**

Disables data collection for the mechanism

**SYNOPSIS**

**C Syntax**

```c
#include <code/anorad.h>
long CxAnoradDisableDataCollection (CxMechanism mech)
```

**ARGUMENTS**

mech     Mechanism ID

**DESCRIPTION**

This function disables all the mechanism axes for data collection. Latched positions are collected from each axis that is enabled for data collection each time the data collection signal goes high, and are rearmed automatically. This eliminates the need for separate clears each time. This function is executed through the mechanism's motion queue. In CIMulation, this function has no effect. The mechanism must be opened for control. Data collection is automatically disabled when a mechanism error is cleared using the CX_MECH_ABORT error action.

**RETURN VALUES**

In C, this function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

In C, if the function returns an error condition, the error code can be obtained by using the CxGetErrorCode function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>Mechanism is not opened for control</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

This example shows how to use the data collection capability.

```c
...
CxMechanism mech;
l long numLeft;
d double values [CX_MAXJOINTS];
...
/*mech has been opened for control prior to this code segment */
CxAnoradEnableDataCollection (mech);
/*data collection signal is triggered here*/
```
CxAnoradDisableDataCollection (mech);

/* retrieve number of positions available (will be non-zero) */
CxAnoradGetDataCollectionLeft (mech, &numLeft);

DxAnoradFlushDataCollection (mech);
/* retrieve number of positions available (will be zero) */
CxAnoradGetDataCollectionLeft (mech, &numLeft)
.
.

SEE ALSO

CxAnoradGetDataCollectionLeft, CxAnoradGetDataCollection,
CxAnoradWaitForDataCollection, CxAnoradEnableDataCollection,
CxAnoradFlushDataCollection
CxAnoradDisableErrorMap
Disables error map

SYNOPSIS
C Syntax

#include <code/anorad.h>
long CxAnoradDisableErrorMap (CxMechanism mech, long map)

ARGUMENTS
mech Mechanism ID
map Map number, from 0 to 9

DESCRIPTION
This function disables an error map to one or two axes. This function is processed through the mechanism's motion queue. The mechanism must be opened for control. The error map must have been previously loaded with the CxAnoradLoadErrorMap function, and enabled with the CxAnoradEnableErrorMap function. This function has no effect in Cimulation.

RETURN VALUES
In C, this function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS
In C, if the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>Mechanism is not opened for control</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>Invalid parameter was passed to this function</td>
</tr>
</tbody>
</table>

EXAMPLE
This example shows how to use the error mapping capability.

```
/* mech has been opened for CX_CONTROL prior to this code segment */

/* load error map */
CxAnoradLoadErrorMap (mech, 1, "OneD.map");
CxAnoradLoadErrorMap (mech, 2, "TwoD.map");

/* enable 1D error map on joint 0 */
CxAnoradEnableErrorMap (mech, 1, 0, -1);

/* disable 1D error map on joint 0 */
CxAnoradDisableErrorMap (mech, 1);
```
/* enable 1D error map on joint1 */
CxAnoradEnableErrorMap (mech, 1, 1, -1);

/* disable 1D error map on joint 1 */
CxAnoradDisableErrorMap (mech, 1);
/* enable 2D error map on joints 0 & 1 */
CxAnoradEnableErrorMap (mech, 2, 0, 1);

/* disable 2D error map */
CxAnoradDisableErrorMap (mech, 2);

/* free maps */
CxAnoradFreeErrorMap (mech, 1);
CxAnoradFreeErrorMap (mech, 2);

SEE ALSO
CxAnoradLoadErrorMap, CxAnoradFreeErrorMap, CxAnoradEnableErrorMap
CxAnoradDwell

Causes the mechanism to pause before continuing

SYNOPSIS

C Syntax

#include <code/anorad.h>
long CxAnoradDwell (CxMechanism mech, long milliseconds)

ARGUMENTS

mech	Mechanism ID
milli-seconds	The time in milliseconds to pause before continuing. The dwell time must be greater than zero.

DESCRIPTION

This function causes the mechanism to maintain its position at a location for the specified time before continuing to the next motion command. This function is processed through the mechanism's motion queue. The mechanism must be opened for control. This function has no effect in CIMulation.

RETURN VALUES

In C, this function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS

In C, if the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Heading</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>Mechanism is not opened for control</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>Invalid parameter was passed to this function</td>
</tr>
</tbody>
</table>

EXAMPLE

This code segment causes the mechanism to pause at position 1 for 500 ms before moving to position 2.

C Syntax

```
CxServer server;
CxMechanism mech;

/* mech has been opened for CX_CONTROL prior to this code segment */
CxMoverAllAxes (mech, position1);
CxAnoradDwell (mech, 500);
CxMoveAllAxes (mech, position2);
```
**CxAnoradEnableDataCollection**

Enables data collection for a mechanism

**SYNOPSIS**

C Syntax

```c
#include <code/anorad.h>
long CxAnoradEnableDataCollection (CxMechanism mech)
```

**ARGUMENTS**

- `mech` Mechanism ID

**DESCRIPTION**

This function enables all the mechanism axes for data collection. Latched positions are collected from each axis enabled for data collection each time the data collection signal goes high. This function is executed through the mechanism's motion queue. In CJMulation, this function has no effect. The mechanism must be opened for control. Data collection is automatically disabled when a mechanism error is cleared with the `CX_MECH_ABORT` error action.

**RETURN VALUES**

In C, this function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

In C, if the function returns an error condition, the error code can be obtained by using the `CxGetErrorNumber` function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>Mechanism is not opened for control</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist</td>
</tr>
</tbody>
</table>

**EXAMPLE**

This example shows how to use the data collection capability.

```c
CxMechanism mech;
long numLeft;
double values[CX_MAXJOINTS];

/* mech has been opened for control prior to this code segment */
CxAnoradEnableDataCollection (mech);

/* retrieve number of positions available */
CxAnoradGetDataCollectionLeft (mech, &numLeft);
for (i = 0; i < numLeft; i++) {
    CxAnoradGetDataCollection (mech, values);
}
```
SEE ALSO

CxAnoradGetDataCollectionLeft, CxAnoradGetDataCollection,
CxAnoradWaitForDataCollection, CxAnoradDisableDataCollection,
CxAnoradFlushDataCollection
CxAnoradEnableErrorMap
Assigns error map to one or two axes

SYNOPSIS

C Syntax

#include <code/anorad.h>
long CxAnoradEnableErrorMap (CxMechanism mech, long map, long joint1, long joint2)

ARGUMENTS

mech     Mechanism ID
map      Map number, from 0 to 9
joint1 Joint number for first axis in error map
joint2 Joint number for second axis in error map

DESCRIPTION

This function assigns an error map to one or two axes. This function is processed through the mechanism's motion
queue. The mechanism must be opened for control. joint1 must be a valid joint number for mech. joint2
must be a valid joint number for mech, or -1 if the error map is only being applied to one axis. The error map
must have been previously loaded with the CxAnoradLoadErrorMap function. This function has no effect in
CIMulation.

RETURN VALUES

In C, this function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS

In C, if the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber
function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>Mechanism is not opened for control</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>Invalid parameter was passed to this function</td>
</tr>
<tr>
<td>CX_INVALID_JOINT_NUMBER</td>
<td>Specified joint number is not valid</td>
</tr>
</tbody>
</table>

EXAMPLE

This example shows how to use the error mapping capability.

    CxMechanism mech;

    /* mech has been opened for CX_CONTROL prior to this code segment */
/* load error map */
CxAnoradLoadErrorMap (mech, 1, "OneD.map");
CxAnoradLoadErrorMap (mech, 2, "TwoD.map");

/* enable 1D error map on joint 0 */
CxAnoradEnableErrorMap (mech, 1, 0, -1);
/* disable 1D error map on joint 0 */
CxAnoradDisableErrorMap (mech, 1);

/* enable 1D error map on joint 1 */
CxAnoradEnableErrorMap (mech, 1, 1, -1);
/* disable 1D error map on joint 1 */
CxAnoradDisableErrorMap (mech, 1);

/* enable 2D error map on joints 0 & 1 */
CxAnoradEnableErrorMap (mech, 2, 0, 1);
/* disable 2D error map */
CxAnoradDisableErrorMap (mech, 2);

/* free maps */
CxAnoradFreeErrorMap (mech, 1);
CxAnoradFreeErrorMap (mech, 2);

SEE ALSO
CxAnoradLoadErrorMap, CxAnoradFreeErrorMap, CxAnoradDisableErrorMap
**CxAnoradEnableJoint**

Enables the amplifier of a single joint

**SYNOPSIS**

**C Syntax**

```c
#include <code/anorad.h>

long CxAnoradEnableJoint (CxMechanism mech, long joint)
```

**ARGUMENTS**

- **mech**  
  Mechanism ID
- **joint**  
  The joint number in the mechanism to enable. Joint numbers start from 0.

**NOTE:** Joint numbers do not necessarily correspond to motor numbers. To determine the joint number refer to the initialization or ".oac" file.

**DESCRIPTION**

This function allows the application developer to enable a single amp or joint at a time.

**NOTE:** Motion cannot occur until all mechanism amplifiers are enabled.

This function is implemented through the CxSendMechanism commend specific for the Anorad IPC motion controller.

**RETURN VALUES**

This function returns **CX_OK (0)** if successful; otherwise, **CX_ERROR (-1)** is returned.

**ERRORS**

If the function returns **CX_ERROR (-1)**, the error code can be obtained by using the CxGetErrrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending a message to the CIMServer</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving a message from the CIMServer</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>The input values are invalid arguments in the function definition</td>
</tr>
</tbody>
</table>

**EXAMPLES**

**C Syntax**

```c
#include <code/robpac.h>
#include <code/anorad.h>

void main( void )
{
    ...  
    /*Open CxServer and CxMechanism */
    ... 
    CxAnoradEnableJoint( Mech, 0 );
}
```
SEE ALSO

CxAnoradDisableJoint
**CxAnoradFlushDataCollection**

Removes all data collection points collected for the mechanism

**SYNOPSIS**

**C Syntax**

```c
#include <code/anorad.h>
long CxAnoradFlushDataCollection (CxMechanism mech)
```

**ARGUMENTS**

mech  Mechanism ID

**DESCRIPTION**

This function flushes all data collected for the mechanism. This function is executed through the mechanism's motion queue. In CIMulation, this function has no effect. The mechanism must be opened for control. To guarantee the data collection buffer remains empty, data collection should be disabled using the `CxAnoradDisableDataCollection` function before flushing the buffer. The data collection buffer is automatically flushed after a mechanism error is cleared using the `CX_MECH_ABORT` error action.

**RETURN VALUES**

In C, this function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

In C, if the function returns an error condition, the error code can be obtained by using the `CxGetErrorNumber` function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>Mechanism is not opened for control</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist</td>
</tr>
</tbody>
</table>

**EXAMPLE**

This example shows how to use the data collection capability.

```c

CxMechanism mech;
long numLeft;
double values [CX_MAXJOINTS];

/* mech has been opened for control prior to this code segment */
CxAnoradEnableDataCollection (mech);

/* data collection signal is triggered here */
CxAnoradDisableDataCollection (mech);
```
/* retrieve number of positions available (will be non-zero) */

CxAnoradGetDataCollectionLeft (mech, &numLeft)

CxAnoradFlushDataCollection (mech);

SEE ALSO

CxAnoradGetDataCollectionLeft, CxAnoradGetDataCollection,
CxAnoradWaitForDataCollection, CxAnoradEnableDataCollection,
CxAnoradDisableDataCollection
CxAnoradFlushStrobeBuffer

Discards all remaining strobe points

SYNOPSIS

C Syntax

```c
#include <code/anorad.h>
long CxAnoradFlushStrobeBuffer (CxMechanism mech)
```

ARGUMENTS

`mech`  Mechanism ID

DESCRIPTION

This function discards all remaining strobe points waiting to be triggered. This function is executed through the mechanism's command queue. In CIMulation, this function has no effect. The mechanism must be opened for control. The strobe buffer is automatically flushed when a mechanism error is cleared using the `CX_MECH_ABORT` error action.

RETURN VALUES

In C, this function returns 0 if successful; otherwise, -1 (`CX_ERROR`) is returned.

ERRORS

In C, if the function returns an error condition, the error code can be obtained by using the `CxGetErrorNumber` function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>Mechanism is not opened for control</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist</td>
</tr>
</tbody>
</table>

EXAMPLE

This example shows how to use the strobe capability.

```c

/* mech has been opened for control prior to this code segment */

CxMoveSingleAxis (mech, 0, 0.0);
CxAnoradSetStrobePoint (mech, 0, 1.0);
CxAnoradSetStrobePoint (mech, 0, 2.0);
CxAnoradSetStrobePoint (mech, 0, 3.0);
CxAnoradFlushStrobeBuffer (mech);
```
/* there should be no strobe points pending */
CxAnoradGetStrobeLeft (mech, &numLeft);

/* no strobing will be triggered */
CxMoveSingleAxis (mech, 0, 4.0);
.
.
SEE ALSO
CxAnoradSetStrobePoint, CxAnoradGetStrobeLeft
CxAnoradFreeErrorMap

SYNOPSIS

C Syntax

#include <code/anorad.h>
long CxAnoradFreeErrorMap (CxMechanism mech, long map)

ARGUMENTS

mech     Mechanism ID
map      Map number to free, from 0 to 9

DESCRIPTION

This function unloads an error map, freeing up the memory associated with the error map. The error map must
have been loaded prior to this call using CxAnoradLoadErrorMap. This function is executed immediately,
and the mechanism must have been opened for control. This function has no effect in CIMulation.

RETURN VALUES

In C, this function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS

In C, if the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber
function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>Mechanism is not opened for control</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>Invalid parameter was passed to this function</td>
</tr>
</tbody>
</table>

EXAMPLE

This example shows how to use the error mapping capability.

    CxMechanism mech;

    /*mech has been opened for CX_CONTROL prior to this code segment */

    /* load error map */
    CxAnoradLoadErrorMap (mech, 1, "OneD.map");
    CxAnoradLoadErrorMap (mech, 2, "TwoD.map");

    /* enable 1D error map on joint 0 */
    CxAnoradEnableErrorMap (mech, 1, 0, -1);
    /* disable 1D error map on joint 0 */
    CxAnoradDisableErrorMap (mech, 1);
/* enable 1D error map on joint 1 */
CxAnoradEnableErrormap (mech, 1, 1, -1);

/* disable 1D error map on joint 1 */
CxAnoradDisableErrorMap (mech, 1);

/* enable 2D error map on joints 0 & 1 */
CxAnoradEnableErrorMap (mech, 2, 0, 1);

/* disable 2D error map */
CxAnoradDisableErrorMap (mech, 2);

/* free maps */
CxAnoradFreeErrorMap (mech, 1);
CxAnoradFreeErrorMap (mech, 2);

SEE ALSO
CxAnoradLoadErrorMap, CxAnoradEnableErrorMap, CxAnoradDisableErrorMap
CxAnoradGetDataCollection
Retrieves data collection point for the mechanism

SYNOPSIS
C Syntax

#include <code/anorad.h>
long CxAnoradDataCollection (CxMechanism mech, double *values)

ARGUMENTS
mech Mechanism ID
values Pointer to array where latched positions will be placed

DESCRIPTION
This function retrieves a data collection point that has been collected for the mechanism. Latched positions are
collected from each axis that is enabled for data collection each time the data collection signal goes high. This
function is executed immediately. In CIMulation, this function will return zeros placed in the values array. The
mechanism may be opened for monitoring or control.

RETURN VALUES
In C, this function returns 0 is successful; otherwise -1 (CX_ERROR) is returned.

ERRORS
In C, if the function returns an error condition, the erro code can be obtained by using the CxGetErrorNumber
function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist</td>
</tr>
</tbody>
</table>

EXAMPLE
This example shows how to use the data collection capability.

```c
CxMechanism mech;
long numLeft;
double values [CX_MAXJOINTS];

/* mech has been opened prior to this code segment */

/* retrieve number of positions available */
CxAnoradGetDataCollectionLeft (mech, &numLeft);
for (i = 0; i < numLeft; i++) {
    CxAnoradGetDataCollection (mech, values);
}
SEE ALSO
CxAnoradGetDataCollectionLeft, CxAnoradWaitForDataCollection,
CxAnoradEnableDataCollection, CxAnoradDisableDataCollection,
CxAnoradFlushDataCollection
CxAnoradGetDataCollectionLeft

Returns the number of data collection points available for reading.

SYNOPSIS
C Syntax

#include <code/anorad.h>
long CxAnoradGetDataCollectionLeft (CxMechanism mech, long *value)

ARGUMENTS
mech Mechanism ID
value Pointer to the variable which is to receive the number of data points left

DESCRIPTION
This function returns the number of data collection points available for reading. Latched positions are collected from each axis enabled for data collection each time the data collection signal goes high. This function is executed immediately. In CIMulation, this function places zero into the variable pointed to by value every time. The mechanism may be opened for monitoring or control.

RETURN VALUES
In C, this function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS
In C, if the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>Invalid parameter was passed to this function</td>
</tr>
</tbody>
</table>

EXAMPLE
This example shows how to use the data collection capability.

```c
CxMechanism mech;
long numLeft;
double values [CX_MAXJOINTS];

/* mech has been opened prior to this code segment */

/* retrieve number of positions available */
CxAnoradGetDataCollectionLeft (mech, &numLeft);
for (i = 0; i < numLeft; I++) {
    CxAnoradGetDataCollection (mech, values);
}
SEE ALSO
  CxAnoradGetDataCollection, CxAnoradWaitForDataCollection,
  CxAnoradEnableDataCollection, CxAnoradDisableDataCollection,
  CxAnoradFlushDataCollection
CxAnoradGetErrorStruct
Retrieves the mechanism error structure

SYNOPSIS
C Syntax
#include <code/anorad.h>
long CxAnoradGetErrorStruct (CxMechanism mech, CxAnoradErrorStruct *errStruct)

ARGUMENTS
mech   Mechanism ID
errStruct Pointer to variable which is to receive error structure

DESCRIPTION
This function retrieves the mechanism error structure. This function is executed immediately. In CIMulation, this function will fill the error structure with zeros. The mechanism may be opened for control or monitor.

RETURN VALUES
In C, this function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS
In C, if the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist</td>
</tr>
</tbody>
</table>

EXAMPLE
This example shows how to retrieve the error structure.

```c
CxMechanism mech;
CxAnoradErrorStruct errStruct;

/* mech has been opened prior to this code segment */
CxAnoradGetErrorStruct (mech, &errStruct);
printf ("Cmd_id: %ld\n", errStruct.cmd_id);
printf ("Timestamp: %lu\n", errStruct.cmd_id);
```
**CxAnoradGetJointStatus**

Returns the motor status for each joint

**SYNOPSIS**

C Syntax

```c
#include <code/anorad.h>
long CxAnoradGetJointStatus (CxMechanism mech, unsigned long *status)
```

**ARGUMENTS**

- `mech`  
  Mechanism ID
- `status`  
  Pointer to array that will be filled with the motor status values

**DESCRIPTION**

This function is executed immediately and fills the status array with the motor status for each joint. The array is filled in joint order and must contain at least as many elements as there are mechanism joints. The mechanism may be opened for monitor or control. In CIMulation, the motor status array will be filled with zeros.

**RETURN VALUES**

In C, this function returns **0** if successful; otherwise, **-1 (CX_ERROR)** is returned.

**ERRORS**

In C, if the function returns an error condition, the error code can be obtained by using the `CxGetErrorNumber` function. The possible error codes are defined in the following table.

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX.MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server</td>
</tr>
<tr>
<td>CX.MESSAGE.Receive_FAILED</td>
<td>Error in receiving message from server</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist</td>
</tr>
</tbody>
</table>

**EXAMPLE**

C Syntax

```c
The following segment gets the joint status for the mechanism Gantry:
long i;
CxMechanism gantry;
Unsigned long js[4];

CxAnoradGetJointStatus (gantry, js);
for (i = 0; i < 4; i++) {
    printf ("Status of joint %d: 0x%x\n", i, js[i]);
}
```
**CxAnoradGetStrobeLeft**

Returns the number of data collection points available for reading

**SYNOPSIS**

C Syntax

```c
#include <code/anorad.h>
long CxAnoradGetStrobeLeft (CxMechanism mech, long *value)
```

**ARGUMENTS**

- `mech` Mechanism ID
- `value` Pointer to the variable which is to receive the number of strobe points left

**DESCRIPTION**

This function returns the number of strobe points waiting to be triggered. This function is executed immediately. In CIMulation, this function places zero into the variable pointed to be value every time. The mechanism may be opened for monitoring or control.

**RETURN VALUES**

In C, this function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

In C, if the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist</td>
</tr>
</tbody>
</table>

**EXAMPLE**

This example shows how to use the strobe capability.

```c

CxMechanism mech;
long numLeft;

/* mech has been opened prior to this code segment */
CxMoveSingleAxis (mech, 0, 0.0);
CxAnoradSetStrobePoint (mech, 0, 1.0);
CxAnoradSetStrobePoint (mech, 0, 2.0);
CxAnoradSetStrobePoint (mech, 0, 3.0);
CxAnoradSetStrobePoint (mech, 0, 5.0);

/* there should be 4 strobe points pending */
CxAnoradGetStrobeLeft (mech, &numLeft);
```
/* strobing will be triggered when joint 0 reaches 1.0, 2.0, and 3.0 */
CxMoveSingleAxis (mech, 0, 4.0);

/* there should be 1 strobe points remaining */
CxAnoradGetStrobeLeft (mech, &numLeft);

SEE ALSO
CxAnoradSetStrobePoint, CxAnoradFlushStrobeBuffer
**CxAnoradGetTrackValue**

Returns the current value of an analog input

**SYNOPSIS**

C Syntax

```c
#include <code/anorad.h>
long CxAnoradGetTrackValue (CxMechanism mech, long joint, double *value)
```

**ARGUMENTS**

- **mech**: Mechanism ID
- **joint**: Joint associated with the analog signal
- **value**: Pointer to the variable which is to receive the value of the analog signal

**DESCRIPTION**

This function returns the value of the analog signal associated with the specified joint. This function is executed immediately. In CIMulation, this function places zero into the variable pointed to by value every time. The mechanism may be opened for monitoring or control.

This function is used mainly for force control, or the focusing of vision systems.

**RETURN VALUES**

In C, this function returns **0** if successful; otherwise, **-1 (CX_ERROR)** is returned.

**ERRORS**

In C, if the function returns an error condition, the error code can be obtained by using the **CxGetErrorNumber** function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist</td>
</tr>
</tbody>
</table>

**EXAMPLE**

This example shows how to use this function.

```c
CxMechanism mech;
double trackValue;

/* mech has been opened prior to this code segment */
CxAnoradGetTrackValue (mech, 2, &trackValue);
printf (Current track value is %f\n", trackValue);
```
**CxAnoradHomeJoint**

Causes specified joint to home

**SYNOPSIS**

C Syntax

```c
#include <code/anorad.h>
long CxAnoradHomeJoint (CxMechanism mech, long joint)
```

**ARGUMENTS**

```
mech       Mechanism ID
joint      Which joint is to be homed
```

**DESCRIPTION**

This function causes the mechanism to home the specified joint. This function is processed through the mechanism's motion queue. The mechanism must be opened for control. This function has no effect in CIMulation.

**RETURN VALUES**

In C, this function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

In C, if the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>Mechanism is not opened for control</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>Invalid parameter was passed to this function</td>
</tr>
</tbody>
</table>

**EXAMPLE**

This code segment causes the mechanism to home the first joint.

C Syntax

```c
CxServer server;
CxMechanism mech;

/* mech has been opened for CX_CONTROL prior to this code segment */
CxAnoradHomeJoint (mech, 0);
```
**CxAnoradLoadErrorMap**

Loads an error map from disk

**SYNOPSIS**

**C Syntax**

```c
#include <code/anorad.h>
long CxAnoradLoadErrorMap (CxMechanism mech, long map, char *filename)
```

**ARGUMENTS**

- **mech** Mechanism ID
- **map** Map number to load, from 0 to 9
- **filename** Name of file containing error map

**DESCRIPTION**

This function loads the specified error map into the specified map number. *filename* must be valid for the file system seen by the server. This file system is not necessarily the same file system seen by the client, if the client and server are on separate computers. Also, because the client and server are separate processes, the current directory of the client may not be the same as the server. This function is processed immediately. The mechanism must be opened for control. This function has no effect in CIMulation.

**RETURN VALUES**

In C, this function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

In C, if the function returns an error condition, the error code can be obtained by using the `CxGetErrorNumber` function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>Mechanism is not opened for control</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>Invalid parameter was passed to this function</td>
</tr>
</tbody>
</table>

**EXAMPLE**

This example shows how to use the error mapping capability.

```c

CxMechanism mech;

/* mech has been opened for CX_CONTROL prior to this code segment */

/* load error map */
CxAnoradLoadErrorMap (mech, 1, "oneD.map");
CxAnoradLoadErrorMap (mech, 2, "TwoD.map");
```
/* enable 1D error map on joint 0 */
CxAnoradEnableErrorMap (mech, 1, 0, -1);

/* disable 1D error map on joint 0 */
CxAnoradDisableErrorMap (mech, 1);

/* enable 1D error map on joint 1 */
CxAnoradEnableErrorMap (mech, 1, 1, -1);

/* disable 1D error map on joint 1 */
CxAnoradDisableErrorMap (mech, 1);

/* enable 2D error map on joints 0 & 1 */
CxAnoradEnableErrorMap (mech, 2, 0, 1);

/* disable 2D error map */
CxAnoradDisableErrorMap (mech, 2);

/* free maps */
CxAnoradFreeErrorMap (mech, 1);
CxAnoradFreeErrorMap (mech, 2);

SEE ALSO
CxAnoradFreeErrorMap, CxAnoradEnableErrorMap, CxAnoradDisableErrorMap
CxAnoradSetStrobePoint
Sets a strobe position for a given axis

SYNOPSIS

C Syntax

#include <code/anorad.h>
long CxAnoradSetStrobePoint (CxMechanism mech, lon joint, double position)

ARGUMENTS

mech Mechanism ID
joint Which joint is associated with this strobe point
position Position at which strobe will be triggered

DESCRIPTION

This function sets a strobe position for the specified axis. Strobe points cause the strobe signal to be pulsed when the specified joint reaches the value given by position. This function is executed through the mechanism's motion queue. In CIMulation, this function has no effect. The mechanism must be opened for control.

RETURN VALUES

In C, this function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS

In C, if the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>Mechanism is not opened for control</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist</td>
</tr>
<tr>
<td>CX_INVALID_JOINT_NUMBER</td>
<td>Specified joint number is not valid</td>
</tr>
</tbody>
</table>

EXAMPLE

This example shows how to use the strobe capability.

```c
CxMechanism mech;

/* mech has been opened for control prior to this code segment */
CxMoveSingleAxis (mech, 0, 0.0);
CxAnoradSetStrobePoint (mech, 0, 1.0);
CxAnoradSetStrobePoint (mech, 0, 2.0);
CxAnoradSetStrobePoint (mech, 0, 3.0);
```
/* strobing will be triggered when joint 0 reaches 1.0, 2.0, and 3.0 */

CxMoveSingleAxis (mech, 0, 4.0);

SEE ALSO

CxAnoradGetStrobeLeft, CxAnoradFlushStrobeBuffer
CxAnoradSetTrack
Sets the analog tracking mode parameters

SYNOPSIS
C Syntax

#include <code/anorad.h>
long CxAnoradSetTrack (CxMechanism mech, long joint, short inrange,
    short value, short error, short clip, short clip2, short start,
    float gain)

ARGUMENTS
mech    Mechanism ID
joint   Joint associated with these tracking mode parameters
inrange Allowable range of analog input, centered on value (In Position Band)
value   Desired analog input value (Target)
error   Maximum range of analog input, centered on value (Following Error)
clip    Maximum tracking speed, from 1 to 1000
clip2   Tracking direction (-1 for down, +1 for up, 0 for both)
start   Allowable range of analog signal to start tracking, centered on value
gain    Multiplier used on analog input, from -1000.0 to 1000.0

DESCRIPTION
This function initializes the parameters for analog tracking mode. The actual tracking operation will not start until
the function CxAnoradTrackOn is called and the analog input is in the start range. This function is processed
through the mechanism's motion queue. The mechanism must be opened for control. This function has no effect in
CIMulation.

RETURN VALUES
In C, this function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

ERRORS
In C, if the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber
function.

The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>Mechanism is not opened for control</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>Invalid parameter was passed to this function</td>
</tr>
</tbody>
</table>
EXAMPLE

The following code segment shows how to use the analog tracking API functions in a force control application.

C Syntax

```c
CxMechanism mech;

/* mech has been opened for CX_CONTROL prior to this code segment */

/* move to clearance point */
CxMoveAllAxes (mech, clearance);

/* set up force control parameters */
CxAnoradSetTrack (mech, joint, 10.0, 1000.0, 2000.0, 50.0, 500.0, 0.75);

/* move down and establish track mode */
CxMoveSingleAxis (mech, joint, hold_position);

/* hold at force for 1 second */
CxAnoradDwell (mech, 1000);

/* change holding force */
CxAnoradSetTrackValue (mech, joint, 1500.0);

/* hold at force for .5 seconds */
CxAnoradDwell (mech, 500);

/* move away */
CxAnoradTrackOff (mech, joint);
CxMoveAllAxes (mech, clearance);
```

SEE ALSO

CxAnoradTrackOn, CxAnoradTrackOff, CxAnoradSetTrackValue
**CxAnoradSetTrackValue**

Sets the desired value of the analog input for tracking mode

**SYNOPSIS**

**C Syntax**

```
#include <code/anorad.h>
long CxAnoradSetTrackValue (CxMechanism mech, long joint, short value)
```

**ARGUMENTS**

- `mech`: Mechanism ID
- `joint`: Joint associated with this tracking mode parameter
- `value`: Desired value for analog input

**DESCRIPTION**

This function initializes the parameters for analog tracking mode. The actual tracking operation will not start until the function `CxAnoradTrackOn` is called and the analog input is in the track range. `CxAnoradTrackOff` is position control mode. This function is processed through the mechanism’s motion queue. The mechanism must be opened for control. This function has not effect in CIMulation.

**RETURN VALUES**

In C, this function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

In C, if the function returns an error condition, the error code can be obtained by using the `CxGetErrorNumber` function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CX_MESSAGE_SEND_FAILED</code></td>
<td>Error in sending message to server</td>
</tr>
<tr>
<td><code>CX_MESSAGE_RECEIVE_FAILED</code></td>
<td>Error in receiving message from server</td>
</tr>
<tr>
<td><code>CX_MECH_NOT_OPEN_FOR_CONTROL</code></td>
<td>Mechanism is not opened for control</td>
</tr>
<tr>
<td><code>CX_INVALID_MECHANISM</code></td>
<td>Invalid mechanism ID, or mechanism does not exist</td>
</tr>
<tr>
<td><code>CX_INVALID_JOINT_NUMBER</code></td>
<td>Specified joint number is not valid</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following code segment shows how to use the analog tracking API functions in a force control application.

**C Syntax**

```c

CxMechanism mech;

/* mech has been opened for CX_CONTROL prior to this code segment */

/*move to clearance point */
CxMoveAllAxes (mech, clearance);

/* set up force control parameters */
```
CxAnoradSetTrack (mech, joint, 10.0, 1000.0, 2000.0, 50.0, 0, 500.0, 0.75)

/* move down and establish track mode */
CxMoveSingleAxis (mech, joint, hold_position);

/* hold at force for 1 second */
CxAnoradDwell (mech, 1000);

/* change holding force */
CxAnoradSetTrackValue (mech, joint, 1500.0);

/* hold at force for .5 seconds */
CxAnoradDwell (mech, 500);

/* move away */
CxAnoradTrackOff (mech, joint);
CxMoveAllAxes (mech, clearance);

SEE ALSO

CxAnoradTrackOn, CxAnoradTrackOff, CxAnoradSetTrack
**CxAnoradTrackOff**

Cancels the analog tracking mode

**SYNOPSIS**

**C Syntax**

```c
#include <code/anorad.h>
long CxAnoradTrackOff (CxMechanism mech, long joint)
```

**ARGUMENTS**

- **mech**  Mechanism ID
- **joint**  Joint associated with the analog tracking mode

**DESCRIPTION**

This function cancels the analog tracking mode, and returns to the position control mode. If the mechanism has not entered tracking mode, the mechanism will not enter track mode until `CxAnoradTrackOn` is called. This function is processed through the mechanism's motion queue. Track mode will be exited if the analog value exceeds the track value or track error. Track mode may also be cancelled with the `CxEnableAmps` or `CxDisableAmps` functions. Track mode will also be cancelled by clearing a mechanism error with the `CX_MECH_ABORT` error action. The mechanism must be opened for control. This function has no effect in CIMulation.

**RETURN VALUES**

In C, this function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

In C, if the function returns an error condition, the error code can be obtained by using the `CxGetErrorNumber` function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>Mechanism is not opened for control</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist</td>
</tr>
<tr>
<td>CX_INVALID_JOINT_NUMBER</td>
<td>Specified joint number is not valid</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following code segment shows how to use the analog tracking API functions in a force control application.

```c
CxMechanism mech;

/* mech has been opened for CX_CONTROL prior to this code segment */
/* move to clearance point */
```
CxMoveAllAxes (mech, clearance);

/* set up force control parameters */
CxAnoradSetTrack (mech, joint, 10.0, 1000.0, 2000.0, 50.0, 0, 500.0, 0.75);
/* move down and establish track mode */
CxMoveSingleAxis (mech, joint, hold_position);

/* hold at force for 1 second */
CxAnoradDwell (mech, 1000);

/* change holding force */
CxAnoradSetTrackValue (mech, joint, 1500.0);

/* hold at force for .5 seconds */
CxAnoradDwell (mech, 500);

/* move away */
CxAnoradTrackOff (mech, joint);
CxMoveAllAxes (mech, clearance);

SEE ALSO

CxAnoradTrackOn, CxAnoradSetTrack, CxAnoradSetTrackValue
**CxAnoradTrackOn**

Turns on analog tracking for a given joint

**SYNOPSIS**

C Syntax

```c
#include <code/anorad.h>
long CxAnoradTrackOn (CxMechanism mech, long joint)
```

**ARGUMENTS**

- `mech`  Mechanism ID
- `joint` Joint on which to enable analog tracking

**DESCRIPTION**

This function enables analog tracking for the specified joint. This function is processed through the mechanism's motion queue. The mechanism must be opened for control. Tracking mode is entered when the analog input is within the tracking range of the tracking value set by the `CxAnoradSetTrackValue` or `CxAnoradSetTrack` functions. After tracking mode has been entered, any additional move commands will be ignored until the tracking mode is cancelled. Track mode will be exited if the analog value exceeds the track value or track error. Track mode may be cancelled with the `CxAnoradTrackOff`, `CxEnableAmps`, or `CxDisableAmps` functions. Track mode will also be cancelled by clearing a mechanism error with the `CX_MECH_ABORT` error action. This function has no effect in Cimulation.

**RETURN VALUES**

In C, this function returns 0 if successful; otherwise, -1 (CX_ERROR) is returned.

**ERRORS**

In C, if the function returns an error condition, the error code can be obtained by using the `CxGetErrorNumber` function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>Mechanism is not opened for control</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist</td>
</tr>
<tr>
<td>CX_INVALID_JOINT_NUMBER</td>
<td>Specified joint number is not valid</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following code segment shows how to use the analog tracking API functions in a force control application.

```c
C Syntax
.
.CxMechanism mech;
.
/* mech has been opened for CX_CONTROL prior to this code segment */
/* move to clearance point */
CxMoveAllAxes (mech, clearance);
```
/* set up force control parameters */
CxAnoradSetTrack (mech, joint, 10.0, 1000.0, 2000.0, 50.0, 0, 500.0, 0.75);

/* move down and establish track mode */
CxMoveSingleAxis (mech, joint, hold_position);

/* hold at force for 1 second */
CxAnoradDwell (mech, 1000);

/* change holding force */
CxAnoradSetTrackValue (mech, joint, 1500.0);

/* hold at force for .5 seconds */
CxAnoradDwell (mech, 500);

/* move away */
CxAnoradTrackOff (mech, joint);
CxMoveAllAxes (mech, clearance);
.
.

SEE ALSO
CxAnoradTrackOff, CxAnoradSetTrack, CxAnoradSetTrackValue
CxAnoradWaitForDataCollection

Waits until a data collection point is collected for the mechanism

SYNOPSIS

C Syntax

#include <code/anorad.h>
l long CxAnoradWaitForDataCollection (CxMechanism mech, double *values, long timeout)

ARGUMENTS

mech     Mechanism ID

DESCRIPTION

This function waits until a data collection point is collected for the mechanism. Latched positions are collected from each axis that is enabled for data collection each time the data collection signal goes high. This function is executed immediately. In CIMulation, this function will return immediately with zeros placed in the values array. The mechanism may be opened for monitoring or control.

RETURN VALUES

This function returns 0 if successful; CX_TIMED_OUT if time expires; -1 if an error occurs.

ERRORS

In C, if the function returns an error condition, the error code can be obtained by using the CxGetErrorNumber function. The possible error codes are defined in the following table:

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_MESSAGE_SEND_FAILED</td>
<td>Error in sending message to server</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>Error in receiving message from server</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>Invalid parameter was passed to this function</td>
</tr>
</tbody>
</table>

EXAMPLE

This example shows how to use the data collection capability.

```
CxMechanism mech;
long numLeft;
double values [CX_MAXJOINTS];

/* mech has been opened for control prior to this code segment */
CxAnoradEnableDataCollection (mech);

/*/wait for data collection, or 1 second, whichever comes first */
CxAnoradWaitForDataCollection (mech, values, 1000);
```
SEE ALSO

CxAnoradGetDataCollectionLeft, CxAnoradGetDataCollection,
CxAnoradEnableDataCollection, CxAnoradDisableDataCollection,
CxAnoradFlushData