- NOTICE -

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 4:18 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

Nodes, Frames and Attributes

long CxAddFeature (CxServer Server, CxNodeId node)
long CxAddGeometry (CxServer Server, CxNodeId node)
long CxAddJoint (CxServer Server, CxNodeId node)
long CxAddNewNode (CxServer Server, const char *name, CxNodeId parent)
long CxAddRegion (CxServer Server, CxNodeId node)
long CxAddRobot (CxServer Server, CxNodeId node)
long CxAddSensor (CxServer Server, CxNodeId node)
long CxAddTcf (CxServer Server, CxNodeId node)
long CxAddTeach (CxServer Server, CxNodeId node)
long CxAddUser (CxServer Server, CxNodeId node)
long CxAttachNode (CxServer Server, CxNodeId child, CxNodeId parent)
long CxClearWorkcell (CxServer Server)
long CxCopyNode (CxServer Server, CxNodeId node, long children_too)
long CxCutNode (CxServer Server, CxNodeId node, long children_too)
long CxDeleteFeature (CxServer Server, CxNodeId node)
long CxDeleteGeometry (CxServer Server, CxNodeId node)
long CxDeleteJoint (CxServer Server, CxNodeId node)
long CxDeleteNode (CxServer Server, CxNodeId node, long children_too)
long CxDeleteRegion (CxServer Server, CxNodeId node)
long CxDeleteRobot (CxServer Server, CxNodeId node)
long CxDeleteSensor (CxServer Server, CxNodeId node)
long CxDeleteTcf (CxServer Server, CxNodeId node)
long CxDeleteTeach (CxServer Server, CxNodeId node)
long CxDeleteUser (CxServer Server, CxNodeId node)
long CxFileDefaultWorkcell (CxServer Server)
long CxGetIgesTol (CxServer Server, double *tol)
long CxGetNamedNodeId (CxServer Server, const char *name, CxNodeId *node)
long CxGetNodeInfo (CxServer Server, CxNodeId node, CxNodeInfo *nodeinfo)
long CxGetNodeName (CxServer Server, CxNodeId node, char *name)
long CxGetNodeRigid (CxServer Server, CxNodeId node, long *rigid_flg)
long CxGetNumberedNodeId (CxServer Server, long number, CxNodeId *node)
long CxGetNumOfNodes (CxServer Server, long *num)
long CxGetSelectedNode (CxServer Server, CxNodeId *node, long *branch)
void CxGetUnit (double *linear_unit_factor, long *angle_unit_flag)
long CxLoadWorkcell (CxServer Server, const char *filename, long ff, 
long operation, CxNodeId merge_at, long geom_flag)
long CxPasteNode (CxServer Server, CxNodeId parent, const char 
*newname)
long CxPickUp (CxServer Server, CxNodeId child, CxNodeId parent)
long CxPlace (CxServer Server, CxNodeId child, CxNodeId parent)
long CxRenameNode (CxServer Server, CxNodeId node, const char *name)
long CxRestoreDefaultState (CxServer Server)
long CxRestoreState (CxServer Server, const char *state_name)
void CxRigidBody1 (CxMatrix t_c_m, double x, double y, double theta, 
CxVector ang, CxVector vec);
void CxRigidBody2 (CxMatrix t_c_m_1, CxMatrix t_c_m_2, CxMatrix 
t_ref2_ref1,double x1, double y1, double x2, double y2, 
CxVector ang, CxVector vec);
long CxSaveDefaultState (CxServer Server)
long CxSaveState (CxServer Server, char *state_name)
long CxSaveWorkcell (CxServer Server, const char *filename, long ff, 
CxNodeId start, long file_branch)
long CxSearchUp (CxServer Server, CxNodeId start_node, unsigned long 
want_mask, long ignore_self, CxNodeId *node_found)
long CxSetIgesTol (CxServer Server, double tol)
long CxSetNodeRigid (CxServer Server, CxNodeId node, long rigid_flg)
long CxSetNodeSelected (CxServer Server, CxNodeId node, long branch, 
long highlight)
void CxSetUnit (double linear_unit_factor, long angle_unit_flag)
CxNodeInfo * CxTraverse (CxServer Server, CxNodeInfo *start, CxNodeInfo 
*node, unsigned long want_mask, unsigned long block_mask, 
CxNodeInfo *node_blk)

Position and Orientation
long CxFindrm (CxServer Server, CxNodeId node, CxNodeId ref_node, 
matrix rel_mat)
long CxFindJntIndepRm (CxServer Server, CxNodeId node, CxNodeId 
ref_node, matrix rel_mat)
long CxGetJntIndepPose (CxServer Server, CxNodeId node, CxNodeId ref, 
char axes [4], CxVector angles, CxVector vec)
long CxGetPose (CxServer Server, CxNodeId node, CxNodeId ref, char axes 
[4], CxVector angles, CxVector vec)
long CxGetRelPose (CxServer Server, CxNodeId node, char axes [4], 
matrix ref_mat)
long CxGetToolPose (CxMechanism mech, CxNodeId tcf, char axes [4], CxVector angles, CxVector vec)
long CxSetDeltaPose (CxServer Server, CxNodeId node, char axes [4], CxVector angles, CxVector vec)
long CxSetJointIndepPose (CxServer Server, CxNodeId node, CxNodeId ref, char axes [4], CxVector angles, CxVector vec)
long CxSetPose (CxServer Server, CxNodeId node, CxNodeId m_frame, CxNode Id meas_ele, char axes [4], CxVector angles, CxVector vec)
long CxSetRelPose (CxServer Server, CxNodeId node, char axes [4], CxVector angles, CxVector vec)
long CxUpdatePose (CxServer Server, CxNodeId node, CxNodeId ref_node, char axes [4], CxVector angles, CxVector vec)

Mechanism Parameters

long CxFileDHParm (CxMechanism mech, CxNodeId node, const char *file_name)
long CxGetAccelTimesMin (CxMechanism mech, CxNodeId node, double *rise_time_min, double *fall_time_min)
long CxGetActualScrewSpeed (CxMechanism mech, CxNodeId node, double *speed)
long CxGetActualToolSpeed (CxMechanism mech, CxNodeId node, double *speed)
long CxSetAccelTimesMin (CxMechanism mech, CxNodeId node, double rise_time_min, double fall_time_min)
long CxSetDefaultSoln (CxMechanism mech, CxNodeId node, long *soln)
long CxSetFclassNumber (CxMechanism mech, CxNodeId node, long class)
long CxSetFwrdkinType (CxMechanism mech, CxNodeId node, long type)
long CxSetIclassNumber (CxMechanism mech, CxNodeId node, long class)
long CxSetInvkinType (CxMechanism mech, CxNodeId node, long type)
long CxSetJclassNumber (CxMechanism mech, CxNodeId node, long jclass)
long CxSetJnt2actMap (CxMechanism mech, long jnt2act_map)
long CxSetMaxScrewSpeed (CxMechanism mech, CxNodeId node, double speed)
long CxSetMaxToolSpeed (CxMechanism mech, CxNodeId node, double speed)
long CxSetRclassNumber (CxMechanism mech, CxNodeId node, long class)
long CxSetScrewAccelMax (CxMechanism mech, CxNodeId node, double accel_rise, double accel_fall)
long CxSetTclassNumber (CxMechanism mech, CxNodeId node, long class)
long CxSetTifFrame (CxMechanism mech, CxNodeId node, CxNodeId frame)
long CxSetTrapAccelMax (CxMechanism mech, CxNodeId node, double accel_rise, double accel_fall)

Axis Parameters

long CxGetJntAccel (CxMechanism mech, long jnt_number, double *accel_rise, double *accel_fall)
long CxGetJntAccel_max (CxMechanism mech, long jnt_number, double *accel_rise_max, double *accel_fall_max)
long CxGetJntDepConst (CxMechanism mech, long dep_jnt_number, long *indep_jnt_number, double *dconst)
long CxGetJntDepType (CxMechanism mech, long jnt_number, long *type)
long CxGetJntInfinite (CxMechanism mech, long jnt_number, long *flag)
long CxGetJntNode (CxMechanism mech, long jnt_number, CxNodeId *node)
long CxGetJntNumber (CxMechanism mech, CxNodeId node, long *jnt_number)
long CxGetJntType (CxMechanism mech, long jnt_number, long *type)
long CxGetMaxDofValue (CxMechanism mech, long jnt_number, double *dof)
long CxGetMaxJntSpeed (CxMechanism mech, long jnt_number, double *speed)
long CxGetMinDofValue (CxMechanism mech, long jnt_number, double *dof)
long CxSetJntAccel (CxMechanism mech, long jnt_number, double accel_rise, double accel_fall)
long CxSetJntAccelMax (CxMechanism mech, long jnt_number, double accel_rise_max, double accel_fall_max)
long CxSetJntDepConst (CxMechanism mech, long dep_jnt_number, long indep_jnt_number, double dconst)
long CxSetJntDepType (CxMechanism mech, long jnt_number, long type)
long CxSetJntInfinite (CxMechanism mech, long jnt_number, long type)
long CxSetJntType (CxMechanism mech, long jnt_number, long type)
long CxSetMaxDofValue (CxMechanism mech, long jnt_number, double dof)
long `CxSetMaxJntSpeed` (CxMechanism mech, long jnt_number, double speed)
long `CxSetMinDofValue` (CxMechanism mech, long jnt_number, double dof)

**Geometry**

long `CxAddCurveSeg` (CxServer Server, CxNodeId node, const char *seg_name, const char *new_seg_name, char axes [4], double a1, double a2, double a3, double x, double y, double z)
long `CxDecompCurveSeg` (CxServer Server, CxNodeId node, const char *seg_name, const char *new_seg_name, long tool_motion_type, long decomp_type, long res_flag, long res, double tol)
long `CxDeleteCurveSeg` (CxServer Server, CxNodeId node, const char *seg_name, const char *next_seg_name)
long `CxGetBoolean` (CxServer Server, CxNodeId node, CxNodeId *node_a, long *operation, CxNodeId *node_b)
long `CxGetBox` (CxServer Server, CxNodeId node, double *L1, double *L2, double *L3)
long `CxGetCone` (CxServer Server, CxNodeId node, double *radius, double *length, long *res_flag, long *res, double *tol)
long `CxGetCurveSegFrame` (CxServer Server, CxNodeId node, const char *seg_name, char axes [4], CxVector angles, CxVector vec)
long `CxGetCurveSegName` (CxServer Server, CxNodeId node, const char *ref_seg_name, char *seg_name, long prev_or_next)
long `CxGetCurveSegRestol` (CxServer Server, CxNodeId node, const char *seg_name, long *res_flag, long *res, double *tol)
long `CxGetCurveSegType` (CxServer Server, CxNodeId node, const char *seg_name, long *type, long *path_axis)
long `CxGetCyl` (CxServer Server, CxNodeId node, double *radius, double *length, long *res_flag, long *res, double *tol)
long `CxGetFrust` (CxServer Server, CxNodeId node, double *R1, double *R2, double *L, long *res_flag, long *res, double *tol)
long `CxGetGeomType` (CxServer Server, CxNodeId node, long *type)
long `CxGetHemi` (CxServer Server, CxNodeId node, double *radius, long *res_flag, long *res, double *tol)
long `CxGetNumCurveSegs` (CxServer Server, CxNodeId node, long *num_segs)
long `CxGetSweepEle` (CxServer Server, CxNodeId node, long *type, double *dist, long *cap_flag, long *res_flag, long *res, double *tol)
long `CxGetSweepFrame` (CxServer Server, CxNodeId node, char axes [4], CxVector angles, CxVector vec)
long `CxGetTrap` (CxServer Server, CxNodeId node, double *L1, double *W1, double *L2, double *W2, double *h)
long `CxMakeBoolean` (CxServer Server, CxNodeId node, CxNodeId node_a, long operation, CxNodeId node_b)
long `CxMakeBox` (CxServer Server, CxNodeId node, double L1, double L2, double L3)
long `CxMakeCone` (CxServer Server, CxNodeId node, double radius, double
long CxMakeCyl (CxServer Server, CxNodeId node, double radius, double
length, long res_flag, long *res, double *tol)

long CxMakeFrust (CxServer Server, CxNodeId node, double R1, double R2,
double L, long res_flag, long *res, double *tol)

long CxMakeHemi (CxServer Server, CxNodeId node, double radius, long
res_flag, long *res, double *tol)

long CxMakeSweepEle (CxServer Server, CxNodeId node, long type, double
dist, long cap_flag, long res_flag, long *res, double *tol)

long CxMakeTrap (CxServer Server, CxNodeId node, double L1, double W1,
double L2, double W2, double h)

long CxSetCurveSegArc (CxServer Server, CxNodeId node, const char
*seg_name, long axis_num)

long CxSetCurveSegArc3 (CxServer Server, CxNodeId node, const char
*seg_name, double x, double y, double z)

long CxSetCurveSegFrame (CxServer Server, CxNodeId node, const char
*seg_name, char axes [4], double ax, double ay, double az, double
x, double y, double z)

long CxSetCurveSegLincirc (CxServer Server, CxNodeId node, const char
*seg_name, long axis_num)

long CxSetCurveSegLinear (CxServer Server, CxNodeId node, const char
*seg_name)

long CxSetCurveSegName (CxServer Server, CxNodeId node, const char
*old_seg_name, const char *new_seg_name)

long CxSetSweepFrame (CxServer Server, CxNodeId node, char axes [4],
double a1, double a2, double a3, double x, double y, double z
Nodes, Frames and Attributes
**CxAddFeature**

Adds the **FEATURE** attribute to a tree node

**SYNOPSIS**

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

**ARGUMENTS**

- **CxServer** The Server ID
- **node** The node ID

**DESCRIPTION**

This function adds the **FEATURE** attribute to a given node in the CIMServer’s workcell file. If a node represents a physical element, which can be measured by sensors, then the **FEATURE** attribute should be set. For example, a feature in the workcell might be a bar code on a part. After creating a node to represent the bar code mark, then add the feature attribute to the 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 Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CX_MESSAGE_SEND_FAILED</strong></td>
<td>An error in sending the message.</td>
</tr>
<tr>
<td><strong>CX_MESSAGE_RECEIVE_FAILED</strong></td>
<td>An error in receiving the message.</td>
</tr>
<tr>
<td><strong>CX_NODE_NOT_FOUND</strong></td>
<td>The given node does not exist.</td>
</tr>
<tr>
<td><strong>CX_MACHINE_OUT_OF_MEMORY</strong></td>
<td>The machine is out of memory.</td>
</tr>
<tr>
<td><strong>CX_NODE_IS_CUT_OUT</strong></td>
<td>The given node has been cut out.</td>
</tr>
<tr>
<td><strong>CX_INVALID_NODE_ID</strong></td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

```c
#include <stdio.h>
#include <code/robpac.h>
void main (void)
{
    CxServer Server;
    CxNodeId world, base_box;

    /* initialize */

    Server = CxOpenServer ("test", CX_SMEM, 0);
    CxGetNamedNodeId (Server, "world", &world);
```
/* add a node named base_box as a child of world */
CxAddNewNode (Server, "base_box", world);

CxGetNamedNodeId (Server, "base_box", &base_box);

/* add feature attribute to node base_box */
CxAddFeature (Server, base_box);

/* make a box of the given dimensions */
if(CxMakeBox (Server, base_box, 100., 200., 300.) == CX_ERROR) {
    fprintf(stderr,"CX_ERROR making base_box...
");
}

/* exit */
CxRobpacExit ();

SEE ALSO

CxDeleteFeature
**CxAddGeometry**

Adds the GEOMETRY attribute to a tree node

**SYNOPSIS**

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

**ARGUMENTS**

- **CxServer**
  - The Server ID
- **Node**
  - The node ID

**DESCRIPTION**

This function adds the GEOMETRY attribute to a given tree node. A tree node must have the GEOMETRY attribute before any geometric shape (box, cylinder, etc.) can be assigned to it.

**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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_MACHINE_OUT_OF_MEMORY</td>
<td>The machine is out of memory.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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 uses CxAddGeometry, as well as two other CODE API functions (CxAddNewNode from the Nodes, Frames & Attributes Section of the CODE API Programmers Reference Manual - Volume 2 and CxMakeBox from the Geometry Section of the CODE API Programmers Reference Manual – Volume 2), to create a box of dimensions 100, 100, and 200. In this case, CxAddGeometry is called to give the GEOMETRY attribute to an existing node (here, the node is created with CxAddNewNode).

```c
CxServer Server;
CxNodeId world, box;
double L1, L2, L3;

/* Define three values for CxMakeBox to use for the sides */
L1 = L2 = 100.0;
L3 = 200.0;

CxGetNamedNodeId (Server, "world", &world);
```
/* add node ele1 as a child of world */
CxAddNewNode (Server, "box", world);

CxGetNamedNodeId (Server, "box", &box);

/* add GEOMETRY attribute to the node */
CxAddGeometry (Server, box);

/* make a box of given dimensions */
CxMakeBox (Server, box, L1, L2, L3);

SEE ALSO

CxDeleteGeometry
**CxAddJoint**

Adds the JOINT attribute to a tree node

**SYNOPSIS**

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

**ARGUMENTS**

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

**DESCRIPTION**

This function adds the JOINT attribute to a given tree node. The JOINT attribute makes it possible for a node to be a joint of a mechanism, and to have all joint related data attached to it. Once a joint is added, it is automatically assigned the default values for a joint. To add the JOINT attribute to a node, the node must be a descendant of a node with the ROBOT attribute. Joints for a mechanism are numbered sequentially beginning at zero.

**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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_MACHINE_OUT_OF_MEMORY</td>
<td>The machine is out of memory.</td>
</tr>
<tr>
<td>CX_NO_ROB_FOR_THE_JOINT</td>
<td>No mechanism is defined for the joint.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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 is part of a program to build a mechanism. In this case, the first joint of the mechanism being created is a child of the mechanism node (which we also create). The function CxAddJoint is used to give “j1” the JOINT attribute.

**NOTE**: Building a mechanism with CIMTools (CODE’s Windows NT user interface) is much easier than writing the source code. The example shown here is an alternative to using the CIMTools interface.

```c
CxServer Server;
CxMechanism mech;
CxNodeId world, robot_base, jnt1;
```
long jnt_number;

/* get node ID for "world" node */
CxGetNamedNodeId (Server, "world", &world)

/* create a node named "robot_base" as a child of the "world" node */
CxAddNewNode Server, "robot_base", world);

CxGetNamedNodeId (Server, "robot_base", &robot_base);

/* add ROBOT attribute to node "robot_base" */
CxAddRobot (Server, robot_base);

/* add joint 1, "j1", as child of "robot_base" mechanism */
CxAddNewNode(Server,"j1", robot_base);

CxGetNamedNodeId (Server, "j1", &jnt1);

/* add JOINT attribute and get the joint number*/
CxAddJoint (Server, jnt1);
CxGetJntNumber (mech, jnt1, &jnt_number);

/* set joint type to translational */
CxSetJntType (mech, jnt_number, CX_TRANS);

SEE ALSO
CxDeleteJoint
CxAddNewNode
Adds a new node to workcell hierarchy

SYNOPSIS
#include <code/robpac.h>
long CxAddNewNode (CxServer Server, char *name, CxNodeId parent)

ARGUMENTS
Server The server ID
name The name of the new node
parent The parent node ID

DESCRIPTION
This function adds a new node to the workcell hierarchy as a child of the parent node, provided that the given parent node exists and the new node name is unique under the given parent. The pose will be the same as that of the parent.

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>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_PARENT_NODE_NOT_FOUND</td>
<td>The given parent node does not exist.</td>
</tr>
<tr>
<td>CX_NAME_CONFLICT</td>
<td>The parent already has a child with the same name.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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_MACHINE_OUT_OF_MEMORY</td>
<td>A memory allocation error.</td>
</tr>
</tbody>
</table>

EXAMPLE
The following example adds a new node named base_box to the world node. The code then adds the GEOMETRY attribute to the new node, and makes a box of dimensions 100, 200, and 300.

#include <stdio.h>
#include <code/robpac.h>
void main (void)
{
    CxServer Server;
    CxNodeId world, base_box;

    /* initialize */
Server = CxOpenServer("test", CX_SMEM, 0);

CxGetNamedNodeId(Server, "world", &world);

/* add a node named base_box as a child of world */
CxAddNewNode(Server, "base_box", world);

CxGetNamedNodeId(Server, "base_box", &base_box);

/* add geometry attribute to node base_box */
CxAddGeometry(Server, base_box);

/* make a box of the given dimensions */
if(CxMakeBox(Server, base_box, 100., 200., 300.) == CX_ERROR) {
    fprintf(stderr,"CX_ERROR making base_box...
"");
}

/* exit */
CxRobpacExit();

SEE ALSO

CxDeleteNode, CxTeachNode
**CxAddRegion**

Adds the REGION attribute to a tree node

**SYNOPSIS**

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

**ARGUMENTS**

- **Server**  
The server ID
- **node**  
The node ID

**DESCRIPTION**

This function adds the REGION attribute to a given tree node. A region defines a physical volume of interest. This function is used to set regions for local calibration.

**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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_MACHINE_OUT_OF_MEMORY</td>
<td>The machine is out of memory.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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 REGION attribute to the workcell node (also called tree node) region1. If successful, the code then sets its region parameters to the following values: Cartesian local volume (clv) to 50 units, mapping pose positional tolerances to 0.1 units, and orientation tolerance to 0.5 degrees. CxSetRegionParm is a calibration API function.

```c
CxServer Server;
CxNodeId region1, world;
CxGetNamedNodeId (Server, "world", world);
CxAddNewNode (Server, "region1", world);
CxGetNamedNodeId (Server, "region1", region1);
```
/ add REGION property to tree node region1 */
if (CxAddRegion (Server, region1) != CX_ERROR) {
    CxSetRegionParm (mech, region1, 50., 0.1, 0.5);
}

SEE ALSO
    CxDeleteRegion
**CxAddRobot**

Adds the ROBOT attribute to a tree node

**SYNOPSIS**

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

**ARGUMENTS**

- **Server**  The server ID
- **node**  The node ID

**DESCRIPTION**

This function adds the ROBOT attribute to a given tree node. The ROBOT attribute makes the node a mechanism by assigning all mechanism related data (kinematics type, velocity settings, etc.) to the node. The workcell must contain at least one node with the ROBOT attribute (a mechanism) for motion to occur.

**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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_MACHINE_OUT_OF_MEMORY</td>
<td>The machine is out of memory.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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**

```
#include <stdio.h>
#include <code/robpac.h>
void main (void)
{
    CxServer Server;
    CxNodeId world, base_box;

    /* initialize */
    Server = CxOpenServer ("test", CX_SMEM, 0);
    CxGetNamedNodeId (Server, "world", &world);

    /* add a node named base_box as a child of world */
```
CxAddNewNode (Server, "base_box", world);

CxGetNamedNodeId (Server, "base_box", &base_box);

/* add robot attribute to node base_box */
CxAddRobot (Server, base_box);

/* make a box of the given dimensions */
if(CxMakeBox (Server, base_box, 100., 200., 300.) == CX_ERROR) {
    fprintf(stderr,"CX_ERROR making base_box...\n");
}

/* exit */
CxRobpacExit ();

SEE ALSO
CxDeleteRobot
CxAddSensor

Adds the SENSOR attribute to a tree node

SYNOPSIS

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

ARGUMENTS

Server     The server ID
node       The node ID

DESCRIPTION

This function adds the SENSOR attribute to a given tree node. Both rigid body updating and mechanism
inaccuracy mapping require sensors. When a sensor, such as a vision camera, is represented by a tree node in
a CODE workcell, the tree node’s SENSOR attribute is set and all sensor-related data is assigned to the 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 Codes</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 given node does not exist.</td>
</tr>
<tr>
<td>CX_MACHINE_OUT_OF_MEMORY</td>
<td>The machine is out of memory.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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

#include <stdio.h>
#include <code/robpac.h>
void main (void)
{
    CxServer Server;
    CxNodeId world, base_box;
    /* initialize */
    Server = CxOpenServer ("test", CX_SMEM, 0);
    CxGetNamedNodeId (Server, "world", &world);
    /* add a node named base_box as a child of world */
CxAddNewNode (Server, "base_box", world);

CxGetNamedNodeId (Server, "base_box", &base_box);

/* add Sensor attribute to node base_box */
CxAddSensor (Server, base_box);

/* make a box of the given dimensions */
if (CxMakeBox (Server, base_box, 100., 200., 300.) == CX_ERROR) {
    fprintf(stderr,"CX_ERROR making base_box...
");
}

/* exit */
CxRobpacExit ();

SEE ALSO
CxDeleteSensor
CxAddTcf

Adds the TCF attribute to a tree node

SYNOPSIS

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

ARGUMENTS

Server  The server ID
node    The node ID

DESCRIPTION

This function adds the TCF attribute to the specified CIMServer tree node. The TCF attribute is used to identify Terminal (or Tool) Control Frames attached to mechanisms. The TCF node must descend from a tree node having the ROBOT attribute; otherwise, an error condition is reported.

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>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 given 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_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
<tr>
<td>CX_TOOL.NOT.ON_ROBOT</td>
<td>The specified node is not attached to a mechanism.</td>
</tr>
</tbody>
</table>

EXAMPLE

```
#include <stdio.h>
#include <code/robpac.h>
void main (void)
{
    CxServer Server;
    CxNodeId world, base_box;

    /* initialize */
    Server = CxOpenServer ("test", CX_SMEM, 0);
    CxGetNamedNodeID (Server, "world", &world);

    /* add a node named base_box as a child of world */
```
CxAddNewNode (Server, "base_box", world);

CxGetNamedNodeId (Server, "base_box", &base_box);

/* add tcf attribute to node base_box */
CxAddTcf (Server, base_box);

/* make a box of the given dimensions */
if(CxMakeBox (Server, base_box, 100., 200., 300.) == CX_ERROR) {
    fprintf(stderr,"CX_ERROR making base_box...
");  
}

/* exit */
CxRobpacExit ();

SEE ALSO
    CxDeleteTcf
**CxAddTeach**

Adds the TEACH attribute to a tree node

**SYNOPSIS**

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

**ARGUMENTS**

- `Server`  The server ID
- `node`    The node ID

**DESCRIPTION**

This function adds the TEACH attribute to a tree node. The TEACH attribute is useful in applications which require frames to be used as pre-taught targets. For example, once a node is taught using a teach pendant, it can be assigned the TEACH attribute. Thereafter, if the user desires to identify pre-taught nodes, the CxTraverse API function can be invoked using the CX_TEACH_MASK.

**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>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 given 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_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

```c
#include <stdio.h>
#include <code/robpac.h>
void main (void)
{
    CxServer Server;
    CxNodeId world, base_box;
    /* initialize */
    Server = CxOpenServer ("test", CX_SMEM, 0);
    CxGetNamedNodeId (Server, "world", &world);
    /* add a node named base_box as a child of world */
    CxAddNewNode (Server, "base_box", world);
```
CxGetNamedNodeId (Server, "base_box", &base_box);

/* add teach attribute to node base_box */
CxAddTeach (Server, base_box);

/* make a box of the given dimensions */
if(CxMakeBox (Server, base_box, 100., 200., 300.) == CX_ERROR) {
    fprintf(stderr,"CX_ERROR making base_box...
");
}

/* exit */
CxRobpacExit ();

SEE ALSO
CxDeleteTeach
**CxAddUser**

Adds the USER attribute to a tree node

**SYNOPSIS**

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

**ARGUMENTS**

- `Server` : The server ID
- `node` : The node ID

**DESCRIPTION**

This function adds the USER attribute to a tree node. The USER attribute is useful in applications in which the user wishes to identify nodes with a specially defined attribute. For example, if the user wanted to define an attribute called MOVABLE_OBJECT, the user would give the node the USER attribute and remember that the node is a MOVABLE_OBJECT. These nodes can then be identified in an application by using the CxTraverse API function with the CX_USER_MASK specified.

**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>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 given 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_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

```c
#include <stdio.h>
#include <code/robpac.h>
void main (void)
{
    CxServer Server;
    CxNodeId world, base_box;

    /* initialize */
    Server = CxOpenServer ("test", CX_SMEM, 0);
    CxGetNamedNodeId (Server, "world", &world);

    /* add a node named base_box as a child of world */
```
CxAddNewNode (Server, "base_box", world);

CxGetNamedNodeId (Server, "base_box", &base_box);

/* add user attribute to node base_box */
CxAddUser (Server, base_box);

/* make a box of the given dimensions */
if(CxMakeBox (Server, base_box, 100., 200., 300.) == CX_ERROR) {
    fprintf(stderr,"CX_ERROR making base_box...\n");
}

/* exit */
CxRobpacExit ();

SEE ALSO

CxDeleteUser
**CxAttachNode**

Attaches one node to another one (changes the parent)

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxAttachNode (CxServer Server, CxNodeId child, CxNodeId parent)
```

**ARGUMENTS**

- **Server**
  - The server ID
- **child**
  - The child node ID (node to be attached)
- **parent**
  - The parent node ID (node to be attached to)

**DESCRIPTION**

This function changes a child’s parent node. It will fail if the given child node and the world node are rigidly attached to the child’s current parent.

The function `CxAttachNode` only changes the logical CIMServer workcell hierarchy. To physically pick up a workcell element with a mechanism’s end effector, you must write code to cause that end effector to take action. If a node is rigidly attached to its current parent, but the current parent is not rigidly attached to its parent, `CxAttachNode` will attach the child node as well as its parent node to the new parent 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 Codes</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 given child node does not exist.</td>
</tr>
<tr>
<td>CX_PARENT_NODE_NOT_FOUND</td>
<td>The given parent node does not exist.</td>
</tr>
<tr>
<td>CX_CANT_PICK_UP_WORLD</td>
<td>The given child node is world, which cannot be picked up.</td>
</tr>
<tr>
<td>CX_CHILD_CANT_PICK_UP_PARENT</td>
<td>The <em>parent</em> is a child of <em>child</em> node.</td>
</tr>
<tr>
<td>CX_NODE_IS_RIGIDLY_ATTACHED</td>
<td>The <em>child</em> node is rigidly attached to its current parent.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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 is from a larger program that drives a mechanism’s moving parts from one location to another. The element `part` is being moved by the vacuum gripper, and the vacuum gripper’s point of action (tool control frame) is represented by `vac_tcf`. 
Note that the actual code to turn the vacuum on and off is separate from the code affecting the workcell hierarchy.

```
    CxServer Server;
    CxMechanism mech;
    CxNodeId part, vac_tcf, table;
    
    /* move vacuum gripper to part */
    CxMoveToNode (mech, part, vac_tcf);
    
    /* pick up "part" using vacuum gripper */
    CxAttachNode (Server, part, vac_tcf);
    
    /* begin code to turn vacuum on in gripper */
    /* end code to turn vacuum on--vacuum is now on and 
    part is attached to gripper */
    
    /* move to assembly table */
    CxMoveToNode (mech, table, vac_tcf);
    
    /* place "part" on the table */
    CxAttachNode (Server, part, table);
    
    /* begin code to turn vacuum off in gripper */
    /* end code to turn vacuum off--vacuum is now off and 
    part is dropped to table top */
    
    /* go on with program */
```

SEE ALSO

CxPickUp, CxPlace
CxClearWorkcell

Clears the CIMServer workcell

SYNOPSIS

#include <code/robpac.h>
long CxClearWorkcell (CxServer Server)

ARGUMENTS

Server    The server ID

DESCRIPTION

This function clears the CIMServer workcell model, such that the resulting workcell has only a world node. This API function will cause any unsaved changes to be lost.

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>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_OTHER_PROCESSES_STILL_RUNNING</td>
<td>This API will not clear the workcell if another process is still connected to the Server.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following example first loads a workcell from the file test.igs in CX_IGES_FORMAT. The workcell file test.igs is in the same directory where this process is running. Another workcell file robot_base.w in C: \ Tom \ cells is then merged with the test.igs at the world node. The file robot_base.w is in CX_ROBLINE_FORMAT. Finally, the function CxClearWorkcell is used to clear the CIMServer workcell model.

```c
CxServer Server;
CxNodeId world;

/* load in an IGES formatted file */
if (CxLoadWorkcell (Server, "test.igs", CX_IGES_FORMAT, CX_OPEN_CELL, world,1)== CX_ERROR) {
    printf ("CX_ERROR in loading workcell test.igs\n");
    CxRobpacExit ();
}

/* merge in a CODE formatted file */
if (CxLoadWorkcell (Server, "C: \ Tom \ cells \ robot_base.w", CX_ROBLINE_FORMAT, CX_MERGE_CELL, world,1) == CX_ERROR) {
    printf("CX_ERROR in merging workcell robot_base.w\n");
}```
    CxRobpacExit ();
}

    /* now clear the workcell */
    CxClearWorkcell (Server);
    .
    .

SEE ALSO

    CxSaveWorkcell, CxLoadWorkcell, CxFileDefaultWorkcell
**CxCopyNode**
Copies a node into a temporary buffer

**SYNOPSIS**

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

**ARGUMENTS**

- **Server**
The server ID
- **node**
The node ID
- **children_too**
The flag (which indicates if the children descending from a node are to be copied (CX_TRUE) or not (CX_FALSE)).

**DESCRIPTION**

This function copies a node from the workcell to a temporary buffer without deleting it from the workcell. If `children_too` is set to CX_TRUE, then the node and all of its children will be copied into the buffer. The information in the buffer can be pasted (see `CxPasteNode`) to a different place in the hierarchy, if desired. The copied node (or branch) remains in the buffer until another call to `CxCopyNode` or `CxCutNode` is made.

**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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_CANT_COPY_WORLD</td>
<td>The given node is world.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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 gets the node IDs. If the DONE_SOLDERING flag is set, it copy’s the branch from the pcb_tray, which was previously attached to the conveyor belt. The branch is pasted to the table so that in simulation it appears as if a pcb_tray is removed from the convoyor and placed on the table.

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

void main (void)
{
    CxServer Server;
    CxMechanism mech;
    CxNodeId MyMech, pcb_tray, conveyor, table, pcb_clips;
```
long done_soldering = 1;

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

CxGetNamedNodeId (Server, "pcb_tray", &pcb_tray);
CxGetNamedNodeId (Server, "pcb_clips", &pcb_clips);
CxGetNamedNodeId (Server, "table", &table);

if (done_soldering)
{
    if (CxCopyNode (Server, pcb_tray, CX_TRUE)!= CX_ERROR)
    {
        CxPasteNode (Server, table, CX_NULL);
    }
}

CxRobpacExit ();

SEE ALSO
    CxCutNode, CxDeleteNode, CxPasteNode
**CxCutNode**

Cuts a node or a branch from the workcell hierarchy

**SYNOPSIS**

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

**ARGUMENTS**

- **Server**
  - The server ID
- **node**
  - The node ID
- **children_too**
  - The flag which indicates whether node or branch operation is desired. If set to `CX_TRUE`, the branch (i.e. the node and its children) referred to by the node will be cut; otherwise, only the named node will be cut from the workcell hierarchy.

**DESCRIPTION**

This function cuts a node (if `children_too` is set to `CX_FALSE`) or a branch (if `children_too` is set to `CX_TRUE`) from the workcell hierarchy, and puts the deleted information in a buffer. If `children_too` is set to `CX_FALSE`, and node has children, then the children of `node` become the children of `node`'s parent. The cut node or branch can then be pasted to another portion of the workcell, if desired (see **CxPasteNode**). It will remain in the buffer until another call to **CxCutNode** or **CxCopyNode** is made. Nodes can be cut and pasted across workcell files, but not across Servers.

**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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_CANT_CUT_WORLD</td>
<td>The world node cannot be deleted.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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_CANT_DELETE_ROBOT</td>
<td>The robot still has dependent joints and therefore may not be deleted. This could cause an adverse condition in the workcell.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following example gets the node IDs. If the DONE_SOLDERING flag is set, it cuts the branch from the pcb_tray, which was previously attached to the conveyor belt. The branch is pasted to the table so that in simulation it appears as if a pcb_tray is removed from the conveyor and placed on the table. It then deletes the pcb_clips node.
#include <code/robpac.h>

void main (void)
{
    CxServer    Server;
    CxMechanism mech;
    CxNodeId   MyMech, pcb_tray, conveyor, table, pcb_clips;
    long       done_soldering = 1;

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

    CxGetNamedNodeId (Server, "pcb_tray", &pcb_tray);
    CxGetNamedNodeId (Server, "pcb_clips", &pcb_clips);
    CxGetNamedNodeId (Server, "table", &table);

    if (done_soldering)
    {
        if (CxCutNode (Server, pcb_tray, CX_TRUE)!= CX_ERROR)
        {
            CxPasteNode (Server, table, CX_NULL);
            CxDeleteNode (Server, pcb_clips, CX_FALSE);
        }
    }
    CxRobpacExit ();
}

SEE ALSO

CxCopyNode, CxDeleteNode, CxPasteNode
**CxDeleteFeature**

Deletes the FEATURE attribute from a tree node

**SYNOPSIS**

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

**ARGUMENTS**

- **Server**
  - The server ID
- **node**
  - The node ID

**DESCRIPTION**

This function deletes the FEATURE attribute from a given tree node. If successful, feature information for that node will be deleted.

**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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The node does not have the FEATURE attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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 <stdio.h>
#include <code/robpac.h>
void main (void)
{
    CxServer Server;
    CxNodeId world, base_box;

    /* initialize */
    Server = CxOpenServer ("test", CX_SMEM, 0);
    CxGetNamedNodeId (Server, "world", &world);

    /* add a node named base_box as a child of world */
    CxAddNewNode (Server, "base_box", world);
```
CxGetNamedNodeId (Server, "base_box", &base_box);
/* delete feature attribute to node base_box */
CxDeleteFeature (Server, base_box);
/* make a box of the given dimensions */
if(CxMakeBox (Server, base_box, 100., 200., 300.) == CX_ERROR) {
   fprintf(stderr,"CX_ERROR making base_box...
");
}
/* exit */
CxRobpacExit ();

SEE ALSO
CxAddFeature
**CxDeleteGeometry**

Deletes the GEOMETRY attribute from a tree node

**SYNOPSIS**

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

**ARGUMENTS**

- **Server**  
  The server ID
- **node**  
  The node ID

**DESCRIPTION**

This function deletes the GEOMETRY attribute from a given tree node. If successful, geometry information for that node will be deleted.

**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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The node does not have the GEOMETRY attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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**

```

CxServer Server;
CxNodeId world, box;
double L1, L2, L3;

/* Define three values for CxMakeBox to use for the sides */
L1 = L2 = 100.0;
L3 = 200.0;

CxGetNamedNodeId (Server, "world", &world);

/* add node ele1 as a child of world */
CxAddNewNode (Server, "box", world);
```
CxGetNamedNodeID (Server, "box", &box);

/* Delete GEOMETRY attribute to the node */
CxDeleteGeometry (Server, box);

/* make a box of given dimensions */
CxMakeBox (Server, box, L1, L2, L3);

SEE ALSO

CxAddGeometry
**CxDeleteJoint**

Deletes the **JOINT** attribute from a tree node

**SYNOPSIS**

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

**ARGUMENTS**

- **Server**
  - The server ID
- **node**
  - The node ID

**DESCRIPTION**

This function deletes the **JOINT** attribute from a given tree node. If successful, **JOINT** information for that node will be deleted.

**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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The node does not have the <strong>JOINT</strong> attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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**

- CxAddJoint
**CxDeleteNode**

Deletes a node or a branch from the workcell hierarchy

**SYNOPSIS**

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

**ARGUMENTS**

- **Server**: The server ID
- **node**: The node ID
- **children_too**: The flag which indicates whether node or branch operation is desired. If set to **CX_TRUE**, the branch (i.e. the node and its children) referred to by the node will be deleted; otherwise, only the named node will be deleted from the workcell hierarchy.

**DESCRIPTION**

This function deletes a node or a branch from the workcell hierarchy. If the value of **CX_TRUE** is passed as the `children_too` parameter, the branch (i.e., the node and all of its children) is deleted. Only the target node will be deleted from the workcell hierarchy if the value of **CX_FALSE** is passed as the `children_too` parameter. If `children_too` is set to **CX_FALSE**, and `node` has children, then the children of `node` become the children of `node`'s parent. Once a node or branch has been deleted, it cannot be pasted back into the workcell hierarchy.

**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><strong>CX_MESSAGE_SENDFAILED</strong></td>
<td>An error in sending the message.</td>
</tr>
<tr>
<td><strong>CX_MESSAGE_RECEIVEFAILED</strong></td>
<td>An error in receiving the message.</td>
</tr>
<tr>
<td><strong>CX_NODE_NOT_FOUND</strong></td>
<td>The given node does not exist.</td>
</tr>
<tr>
<td><strong>CX_CANT_CUT_WORLD</strong></td>
<td>The <code>world</code> node cannot be deleted.</td>
</tr>
<tr>
<td><strong>CX_NODE_IS_CUT_OUT</strong></td>
<td>The given node has been cut out.</td>
</tr>
<tr>
<td><strong>CX_INVALID_NODE_ID</strong></td>
<td>The node number does not match the given node ID number.</td>
</tr>
<tr>
<td><strong>CX_CANT_DELETE_ROBOT</strong></td>
<td>The robot node still has dependent joints.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following example gets the node IDs. If the **DONE_SOLDERING** flag is set, it cuts the branch from the `pcb_tray`, which was previously attached to the conveyor belt. It pastes that branch to the table so that in simulation it appears as if a `pcb_tray` is removed from the conveyor and placed on the table. It then deletes the `pcb_clips` node. After that, the `pcb_clips` can no longer be accessed.

```c
#include <code/robpac.h>
```
void main(void)
{
    CxServer    Server;
    CxMechanism mech;
    CxNodeId   MyMech, pcb_tray, conveyor, table, pcb_clips;
    long      done_soldering = 1;

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

    CxGetNamedNodeId (Server, "pcb_tray", &pcb_tray);
    CxGetNamedNodeId (Server, "pcb_clips", &pcb_clips);
    CxGetNamedNodeId (Server, "table", &table);

    if (done_soldering)
    {
        if (CxCutNode (Server, pcb_tray, CX_TRUE)!= CX_ERROR)
        {
            CxPasteNode (Server, table, CX_NULL);
            CxDeleteNode (Server, pcb_clips, CX_FALSE);
        }
    }

    CxRobpacExit();
}

SEE ALSO
    CxCopyNode, CxCutNode, CxPasteNode
**CxDeleteRegion**

Deletes the REGION attribute from a tree node

**SYNOPSIS**

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

**ARGUMENTS**

- **Server**  The server ID
- **node**  The node ID

**DESCRIPTION**

This function deletes the REGION attribute from a given tree node. If successful, all region information for that node will be deleted.

**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>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 given 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_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>The node does not have the REGION attribute.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

CxAddRegion
CxDeleteRobot

Deletes the ROBOT attribute from a tree node

SYNOPSIS

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

ARGUMENTS

- Server: The server ID
- node: The node ID

DESCRIPTION

This function deletes the ROBOT attribute from a given tree node. If successful, all mechanism information for that node will be deleted.

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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_CANT_DELETE_ROBOT</td>
<td>The robot still has dependent joints and therefore may not be deleted. This could cause an adverse condition in the workcell.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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>The node does not have the ROBOT attribute.</td>
</tr>
</tbody>
</table>

SEE ALSO

CxAddRobot
CxDeleteSensor

Deletes the SENSOR attribute from a tree node

SYNOPSIS

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

ARGUMENTS

- **Server**: The server ID
- **node**: The node ID

DESCRIPTION

This function deletes the SENSOR attribute from a given tree node. If successful, all sensor information stored in the node will be deleted.

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>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 given 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_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>The node does not have the SENSOR attribute.</td>
</tr>
</tbody>
</table>

SEE ALSO

 CáxAddSensor
**CxDeleteTcf**

Deletes the TCF attribute from a tree node

**SYNOPSIS**

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

**ARGUMENTS**

- **Server**  
  The server ID
- **node**  
  The node ID

**DESCRIPTION**

This function deletes the TCF attribute 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 `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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The index number does not match the node ID number.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The given node has been cut out.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the TCF attribute.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

CxAddTcf
**CxDeleteTeach**

Deletes the TEACH attribute from a tree node

**SYNOPSIS**

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

**ARGUMENTS**

- **Server**  
The server ID
- **node**  
The node ID

**DESCRIPTION**

This function deletes the TEACH attribute 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 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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The index number does not match the node ID number.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The given node has been cut out.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the TEACH attribute.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

CxAddTeach
**CxDeleteUser**

Deletes the **USER** attribute from a tree node

**SYNOPSIS**

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

**ARGUMENTS**

- **Server**
  - The server ID
- **node**
  - The node ID

**DESCRIPTION**

This function deletes the **USER** attribute 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 `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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The index number does not match the node ID number.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The given node has been cut out.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the <strong>USER</strong> attribute.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

CxAddUser
**CxFileDefaultWorkcell**

Saves workcell to the default workcell file

**SYNOPSIS**

```
#include <code/robpac.h>
long CxFileDefaultWorkcell (CxServer Server)
```

**ARGUMENTS**

- **Server**
  - The server ID

**DESCRIPTION**

This function saves a workcell to the same workcell file that has been loaded by the Server, but only works if a workcell in `CX_ROBLINE_FORMAT` has been loaded. You must also have write permission for the file to be saved.

**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><code>CX_MESSAGE_SEND_FAILED</code></td>
<td>An error in sending the message.</td>
</tr>
<tr>
<td><code>CX_MESSAGE_RECEIVE_FAILED</code></td>
<td>An error in receiving the message.</td>
</tr>
<tr>
<td><code>CX_FILE_OPEN_ERROR</code></td>
<td>The default workcell file cannot be opened for writing.</td>
</tr>
<tr>
<td><code>CX_NO_DEFAULT_WORKCELL</code></td>
<td>The default workcell file does not exist.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following example opens a connection to the Server and loads a workcell. It then modifies the workcell, and saves it back to the same file.

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

void main(void)

    CxServer Server;
    CxNodeId world;

    /* open the server */
    Server = CxOpenServer ("Testing", CX_SMEM, 0);

    CxLoadWorkcell (Server, "RT2.w", CX_ROBLINE_FORMAT, CX_OPEN_CELL, world, 1);

    /* Modify workcell, adding geometry, mechanisms, etc */
    ...

    /* save workcell to the default workcell file */
```
CxFileDefaultWorkcell(Server);

CxRobpacExit();

SEE ALSO

CxSaveWorkcell, CxLoadWorkcell
**CxGetIgesTol**

Gets the IGES conversion tolerance

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxGetIgesTol (CxServer Server, double *tol)
```

**ARGUMENTS**

- **Server**
  - The Server ID
- **tol**
  - The IGES conversion tolerance

**DESCRIPTION**

This function gets the IGES conversion tolerance.

IGES (Initial Graphics Exchange Specification) uses an arc center point, a starting point, and an end point to represent a circle or circular arc. The CODE database, on the other hand, uses edges and polygons to approximate a circle and a cylinder. The number of edges used to create the element is referred to as the resolution. Tolerance refers to the difference between the radius of the true circle, and the minimum radius of the polygonalized circle. The radius and tolerance, or the resolution can be used to determine the edges and polygons.

The higher the resolution or the lower the tolerance, the closer the approximation is to a true circle. Resolution and tolerance are interdependent. Once a resolution is given for a circle with known radius, the tolerance is calculated, and vice versa. IGES tolerance is the tolerance used to convert an IGES circular entity into the CIMServer geometry format. The following figure depicts the relationship between resolution, tolerance, and radius.

```
res = 6
```

**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>An error in sending the message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>An error in receiving the message.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

CxSetIgesTol
CxGetNamedNodeId

Gets the node ID for a given named node

SYNOPSIS

#include <code/robpac.h>
long CxGetNamedNodeId (CxServer Server, char *name, CxNodeId *node)

ARGUMENTS

Server        The Server ID
name          The name of the node (maximum 255 characters)
node          The returned node ID

DESCRIPTION

This function gets the node ID (node number and index) from the given name of a node, and assigns these
values to the variable node. This function does not allocate any memory for node. The calling side must
allocate memory for it.

One unique feature about this function is how the node name can be specified for nodes of interest. When a
node name alone is not sufficient to define the node of interest in the workcell, a search path can be specified
to help identify the node. The CIMServer recognizes the child search symbol “/”, and traverse search symbol
“\.” The child search symbol (/) is used to specify the absolute path to a node, i.e., “/robot1/j1/j2/j3.”
The traverse search symbol (^) is used to specify that the node of interest is a descendant of the given parent
node, i.e., “robot1^j3.”

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>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

The following example will get the CxNodeId for a tree node named peg.

CxServer Server;
CxNodeId peg;

CxGetNamedNodeId(Server, "peg", &peg);
When an existing workcell is merged with another workcell, some of the node names may be duplicated. The most commonly seen names include j1, j2, j3, tcf, box, cylinder, etc. To get the CxNodeId of j4 of the robot_base robot and the j4 of the robot2_base robot, the following two methods can be used: */

CxGetNamedNodeId (Server, "robot_base/j1/j2/j3/j4", &robot_base_j4);
/* —or— */
CxGetNamedNodeId (Server, "robot2_base^j4", robot3_base_j4);

SEE ALSO
CxGetNumberedNodeId, CxGetNodeInfo
**CxGetNodeInfo**

Gets the node information for a given node ID

**SYNOPSIS**

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

**ARGUMENTS**

- **Server**  The server ID
- **node**  The node ID
- **nodeinfo**  The returned node information

**DESCRIPTION**

This function gets node information from a given node ID. This function does not allocate any memory for CxNodeInfo. The calling application must provide the memory for the nodeinfo parameter. (Declare variable of type CxNodeInfo).

CxNodeInfo is defined in `install_dir/include/code/robpac_defs.h` as:

```c
typedef struct CxNodeInfo
{
  long number;
  long node_index;
  char name[CX_MAXNAME];
  unsigned long mask;
  CxNodeId child;
  CxNodeId parent;
  CxNodeId older;
  CxNodeId younger;
} CxNodeInfo;
```

This function is useful when more information about a tree node is desired, for example: what attributes does it have, children, siblings, who is the parent, etc…

**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>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 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_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>
EXAMPLE

The following example gets the child node name of a given node named ele.

```c
cxserver server;
cxnodeid ele_id;
cxnodeinfo ele_info;
char child_name[CX_MAXNAME];

CxGetNamedNodeId(server, "ele", &ele_id);
CxGetNodeInfo(server, ele_id, &ele_info);

/* check if child node exists */
if (ele_info.child.nodeindex != 0)
{
    CxGetNodeName(server, ele_info.child, child_name);
    printf("The first child of ele is: %s\n", child_name);
}
else
{
    printf("Node ele does not have a child node\n");
}

SEE ALSO

CxGetNamedNodeId, CxGetNumberedNodeID
```
CxGetNodeName

Gets the node name for a given node ID

SYNOPSIS

#include <code/robpac.h>
long CxGetNodeName (CxServer Server, CxNodeId node, char *name)

ARGUMENTS

Server   The server ID
node     The node ID
name     The name of the node

DESCRIPTION

This function gets the name of a node with a specified node ID (number and index).

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>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 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_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following example gets the node ID of the selected node and prints out that name if the selected node is valid.

```
CxServer Server;
CxNodeId node1;
char node_name[CX_MAXNAME];
Long branch_selected;

/* Get ID of the selected node */
CxGetSelectedNode (Server, &node1, &branch_selected);

/* If selected node is a valid node print its name */
if  (CxGetNodeName (Server, node1, node_name) != CX_ERROR)
{
   printf ("Selected node name is : %s\n", node_name);
```
SEE ALSO

CxGetNamedNodeId, CxGetNumberedNodeId
CxGetNodeRigid

Gets the node’s rigid attachment status

SYNOPSIS

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

ARGUMENTS

- **Server** The Server ID
- **node** The node ID
- **rigid_flg** A flag which sets whether or not the node is rigidly attached to its parent (CX_TRUE or CX_FALSE)

DESCRIPTION

This function gets the node rigid status. When a node is rigidly attached to its parent, you cannot pick the node up by itself. When you pick up a node that is rigidly attached to its parent, both the node and its parent will be picked up. In other words, you cannot change the parent of a node that is rigidly attached to its parent (you can change the parent of the node if it is not rigidly attached).

When a node is created, the default rigid status is set to CX_TRUE (rigidly attached).

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>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 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_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following example checks the rigid status for node `peg` before picking it up with `gripper`. If the node `peg` is rigidly attached to its parent, its status is set to CX_FALSE.

```c
CxServer Server;
CxNodeId peg, gripper;
long rigidlyAttached;

/* check if peg is rigidly attached to its parent */
CxGetNodeRigid (Server, peg, &rigidlyAttached);
if (rigidlyAttached)
{
    CxSetNodeRigid (Server, peg, CX_FALSE);
}
```
/* pick up the peg */
CxPickUp (Server, peg, gripper)

SEE ALSO
  CxSetNodeRigid
**CxGetNumberedNodeId**

Gets the node ID for a given node number

**SYNOPSIS**

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

**ARGUMENTS**

- **Server**: The server ID
- **number**: The node number
- **node**: The node ID

**DESCRIPTION**

This function gets a node ID of a node with a given number.

**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>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 numbered node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The given node has been cut out.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following example gets the name of the node whose unique number is “2.” It uses CxGetNumberedNodeId to get the CxNodeId of the required node.

```c
    CxServer Server;
    CxNodeId node2;
    char node2_name[CX_MAXNAME];
    ...
    CxGetNumberedNodeId (Server, 2, &node2);
    CxGetNodeName (Server, node2, node2_name);
    ...
```

**SEE ALSO**

CxGetNamedNodeId, CxGetNodeInfo
CxGetNumOfNodes

Gets the total number of nodes currently loaded in the workcell

SYNOPSIS

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

ARGUMENTS

Server The Server ID
num The total number of nodes loaded in the workcell

DESCRIPTION

This function gets the total number of nodes currently loaded in the workcell.

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>An error in sending the message.</td>
</tr>
<tr>
<td>CX.MESSAGE_RECEIVE_FAILED</td>
<td>An error in receiving the message.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following example gets the total number of nodes present in a workcell and prints it out. Next, it adds a new node to the workcell, then gets the new number of nodes and prints it out. Assuming that the node gets added successfully, the total number of nodes is incremented.

```
CxServer Server;
long   total_nodes;

/* display total number of nodes before adding a node */
CxGetNumOfNodes (Server, &total_nodes);
printf ("Currently total number of nodes is: %ld\n", total_nodes);
CxAddNewNode (Server, "box", world);

/* After adding a node total nodes will be one greater */
/* than previous total number */
CxGetNumOfNodes (Server, &total_nodes);
printf("Now the total number of nodes is: %ld\n", total_nodes);
```

CxGetSelectedNode

Gets the node ID of last selected node in the workcell

SYNOPSIS

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

ARGUMENTS

- **Server**: The server ID
- **node**: The node ID
- **branch**: The flag which determines whether the entire branch (node and its children) is selected (CX_TRUE), or just a single node is selected (CX_FALSE)

DESCRIPTION

This function gets the node ID of the most recently selected node in the workcell. If no node has been selected, the node ID of the `world` node is returned. A node is selected by picking it from CIMulation's graphical screen using the mouse cursor and pressing the left mouse button, or by using the API function `CxSetNodeSelected`.

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>An error in sending the message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>An error in receiving the message.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following example gets the currently selected node, and prints out the name of the node. It also checks if the branch is selected.

```c
CxServer Server;
char name[CX_MAXNAME];
CxNodeId node;
long branch;

if (CxGetSelectedNode (Server, &node, &branch) == CX_ERROR)
{
    printf ("No node has been selected.\n");
}
else
{
    CxGetNodeName (Server, node, name);
    printf ("You have just selected: %s\n", name);
}
if (branch)
{
    printf ("Branch has been selected.\n");
}
else
{
    printf ("Branch has not been selected.\n");
}

SEE ALSO

CxSetNodeSelected
**CxGetUnit**

Gets the current linear unit conversion factor and the unit used for angular quantities

**SYNOPSIS**

```c
#include <code/robpac.h>
void CxGetUnit (double *linear_unit_factor, long *angle_unit_flag)
```

**ARGUMENTS**

- `linear_unit_factor`: The unit conversion factor for linear quantities
- `angle_unit_flag`: The unit used for angular quantities. Its value can be `CX_USE_DEGREE` which is the default setting, or `CX_USE_RADIANS`.

**DESCRIPTION**

This function gets the current linear conversion factor and the unit used for angular quantities.

Linear quantities used in various API functions are multiplied by the `linear_unit_factor` when passed from a CODE application process to the CIMServer, and divided by the `linear_unit_factor` when passed in the other direction. The default value of `linear_unit_factor` is 1.0. This implies that linear values are sent without any change between the CODE application process and the CIMServer.

The unit for angular quantities can be either `CX_USE_DEGREE` or `CX_USE_RADIANS`. The CIMServer always uses radians for angular quantities. Therefore, if a CODE application process sets its angular unit to `CX_USE_DEGREE`, angular values will be converted to radians before being sent to the server. Also, values returned by the Server will be converted from radians to degree before being returned to the process. If the process has its angular unit flag set to `CX_USE_RADIANS`, the value will be passed between the client and the server without any conversion. The default value of `angle_unit_flag` is `CX_USE_DEGREE`.

**RETURN VALUES**

This function does not return any values.

**EXAMPLE**

The following example gets the current unit settings and prints out to the screen.

```c
double linear_unit_factor, long angle_unit_flag;

CxGetUnit (&linear_unit_factor, &angle_unit_flag);
printf (" unit conversion factor = %lf\n", linear_unit_factor);
if (angle_unit_flag == CX_USE_RADIAN)
  printf ("Radians are used for angular quantities);  
else
  printf ("Degrees are used for angular quantities);  
```

**SEE ALSO**

`CxSetUnit`
**CxLoadWorkcell**

Loads a workcell from a file

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxLoadWorkcell(CxServer Server, char *filename, long ff, long operation, CxNodeId merge_at, long geom_flag)
```

**ARGUMENTS**

- **Server**
  The Server ID

- **filename**
  The name of the file to be loaded by the CIMServer. For non-RTX CODE software, the directory of the filename is where the Server is started. For RTX CODE software, the directory is the %SystemRoot%System32 directory.

- **ff**
  File format: CX_ROBLINE_FORMAT (the standard CIMServer format), CX_IGES_FORMAT. The file format constants are defined in <code/robconst.h>

- **operation**
  The loading operation. The choices are whether to merge the workcell (CX_MERGE_CELL), or open the workcell (CX_OPEN_CELL). These constants are defined in <code/robconst.h>

- **merge_at**
  The node ID merge location in the hierarchy tree from the file. The merge_at node ID is only used for the CX_MERGE_CELL operation. It is ignored when the operation is CX_OPEN_CELL.

- **geom_flag**
  The geometry flag which determines whether geometry information will be loaded (with geometry = 1, without geometry = 0).

**DESCRIPTION**

This function loads a workcell from a file, or merges a file with the existing workcell at a specified merge location (merge_at). The workcell file may be either in CX_ROBLINE_FORMAT, or in CX_IGES_FORMAT. The user may load the workcell with or without geometry.

**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_INVALID_OPERATION</td>
<td>The operation is not defined. CX_MERGE_CELL or CX_OPEN_CELL are required.</td>
</tr>
<tr>
<td>CX_FILE_OPEN_ERROR</td>
<td>The named workcell file cannot be opened.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>The named merge_at node does not exist for the CX_MERGE_CELL operation.</td>
</tr>
<tr>
<td>CX_MISMATCHED_FILE_FORMAT</td>
<td>The specified file format does not match the format of the existing file.</td>
</tr>
<tr>
<td>CX_FILE&gt;Loading_ERROR</td>
<td>A file loading error.</td>
</tr>
<tr>
<td>CX_OTHER_PROCESS_STILL_RUNNING</td>
<td>A workcell cannot be opened if there are other processes still running.</td>
</tr>
<tr>
<td>Error Codes</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>CX_INVALID_FILE_FORMAT</td>
<td>The specified file format is not valid.</td>
</tr>
<tr>
<td>CX_NEWER_WORKCELL_FILE_VERSION</td>
<td>The workcell file version is newer than what the server supports.</td>
</tr>
<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_MISMATCHED_IGES_FILE_FORMAT</td>
<td>The specified cell is not in IGES format.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following example loads a workcell from file “test.w” with CX_ROBLINE_FORMAT. The workcell file “test.w” is in the directory where this process is started.

```c
if (CxLoadWorkcell(Server, "test.w", CX_ROBLINE_FORMAT, CX_OPEN_CELL, 1)== CX_ERROR)
{
    printf("CX_ERROR in loading workcell test.w\n");
}
```

If a full path is included, the workcell file will be loaded from the specified directory. The following statement will load the workcell “test.w” from C:\Tom\cells.

```c
CxLoadWorkcell(Server, "C:\Tom\cells\test.w", CX_ROBLINE_FORMAT, CX_OPEN_CELL, world, 1);
```

**SEE ALSO**

CxSaveWorkcell, CxClearWorkcell, CxFileDefaultWorkcell
**CxPasteNode**

Pastes a node or branch into a specified location in the workcell hierarchy

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxPasteNode(CxServer Server, CxNodeId parent, char *newname)
```

**ARGUMENTS**

- **Server**
  The server ID

- **Parent**
  The Id of the node which will become the new parent of the node in the buffer pasted under parent. The name of the pasted node will be changed to `new_name`.

- **newname**
  If specified, the node will be pasted; otherwise, the original name will be used if `(char *) CX_NULL` is given.

**DESCRIPTION**

This function pastes a previously copied or cut node or branch (i.e., a node and its children) from a temporary buffer to a location specified by `parent` in the workcell file hierarchy. Pasting the node does not change its pose relative to its `parent`. It also does not zero out the position or orientation of the node. Specifying a new name will change the node name.

**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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_EMPTY_BUFFER</td>
<td>Nothing is in the buffer to paste.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following example cuts the robot `s100` and pastes it to `table`. The robot’s name changes to `GMFs100`, and it becomes a child of `table`.

```c
CxCutNode (Server, s100, CX_TRUE);
CxPasteNode (Server, table, "GMFs100");
```

**SEE ALSO**

CxCopyNode, CxCutNode, CxDeleteNode
**CxPickUp**

Picks up an object (changes the parent)

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxPickUp(CxServer Server, CxNodeId child, CxNodeId parent)
```

**ARGUMENTS**

- **Server**  
The server ID
- **child**  
The child node ID
- **parent**  
The parent node ID

**DESCRIPTION**

This function changes a child’s parent node. It will fail if the given child node and the world node are rigidly attached to the child’s current parent.

The function `CxAttachNode` only changes the logical CIMServer workcell hierarchy. To physically pick up a workcell element with a mechanism’s end effector, you must write code to cause that end effector to take action. If a node is rigidly attached to its current parent, but the current parent is not rigidly attached to its parent, `CxAttachNode` will attach the child node as well as its parent node to the new parent 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 Codes</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 given child node does not exist.</td>
</tr>
<tr>
<td>CX_PARENT_NODE_NOT_FOUND</td>
<td>The given parent node does not exist.</td>
</tr>
<tr>
<td>CX_CANT_PICK_UP_WORLD</td>
<td>The given child node is world, which cannot be picked up.</td>
</tr>
<tr>
<td>CX_CHILD_CANT_PICK_UP_PARENT</td>
<td>The parent is a child of child node.</td>
</tr>
<tr>
<td>CX_NODE_IS_RIGIDLY_ATTACHED</td>
<td>The child node is rigidly attached to its current parent.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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 is from a larger program that drives a mechanism’s moving parts from one location to another. The element *part* is being moved by the vacuum gripper, and the vacuum gripper’s point of action (tool control frame) is represented by `vac_tcf`. 
Note that the actual code to turn the vacuum on and off is separate from the code affecting the workcell hierarchy.

```c
CxServer Server;
CxMechanism mech;
CxNodeId part, vac_tcf, table;

/* move vacuum gripper to part */
CxMoveToNode (mech, part, vac_tcf);

/* pick up "part" using vacuum gripper */
CxAttachNode (Server, part, vac_tcf);

/* begin code to turn vacuum on in gripper */

/* end code to turn vacuum on--vacuum is now on and part is attached to gripper */

/* move to assembly table */
CxMoveToNode (mech, table, vac_tcf);

/* place "part" on the table */
CxAttachNode (Server, part, table);

/* begin code to turn vacuum off in gripper */

/* end code to turn vacuum off--vacuum is now off and part is dropped to table top */

/* go on with program */
```

**SEE ALSO**

CxPickUp, CxPlace
CXPlace
Places an object (change parent)

SYNOPSIS
#include <code/robpac.h>
long CXPlace(CxServer Server, CxNodeId child, CxNodeId parent)

ARGUMENTS
Server      The server ID
child      The child node ID
parent      The parent node ID

DESCRIPTION
This function changes a child’s parent node. It will fail if the given child node and the world node are rigidly
attached to the child’s current parent.

The function CXAttachNode only changes the logical CIMServer workcell hierarchy. To physically pick
up a workcell element with a mechanism’s end effector, you must write code to cause that end effector to take
action. If a node is rigidly attached to its current parent, but the current parent is not rigidly attached to its
parent, CXAttachNode will attach the child node as well as its parent node to the new parent 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 Codes</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 given child node does not exist.</td>
</tr>
<tr>
<td>CX_PARENT_NODE_NOT_FOUND</td>
<td>The given parent node does not exist.</td>
</tr>
<tr>
<td>CX_CANT_PICK_UP_WORLD</td>
<td>The given child node is world, which cannot be picked up.</td>
</tr>
<tr>
<td>CX_CHILD_CANT_PICK_UP_PARENT</td>
<td>The parent is a child of child node.</td>
</tr>
<tr>
<td>CX_NODE_IS_RIGIDLY_ATTACHED</td>
<td>The child node is rigidly attached to its current parent.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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 is from a larger program that drives a mechanism’s moving parts from one location to
another. The element part is being moved by the vacuum gripper, and the vacuum gripper’s point of action
(tool control frame) is represented by vac_tcf.

Note that the actual code to turn the vacuum on and off is separate from the code affecting the workcell hierarchy.
CxServer Server;
CxMechanism mech;
CxNodeId part, vac_tcf, table;

/* move vacuum gripper to part */
CxMoveToNode (mech, part, vac_tcf);

/* pick up "part" using vacuum gripper */
CxAttachNode (Server, part, vac_tcf);

/* begin code to turn vacuum on in gripper */
/* end code to turn vacuum on--vacuum is now on and part is attached to gripper */

/* move to assembly table */
CxMoveToNode (mech, table, vac_tcf);

/* place "part" on the table */
CxAttachNode (Server, part, table);

/* begin code to turn vacuum off in gripper */
/* end code to turn vacuum off--vacuum is now off and part is dropped to table top */

/* go on with program */

SEE ALSO
CxPickUp
**CxRenameNode**

Renames a node in the workcell

**SYNOPSIS**

```
#include <code/robpac.h>
long CxRenameNode (CxServer Server, CxNodeId node, char* name)
```

**ARGUMENTS**

- **Server**
  The Server ID
- **node**
  The node ID
- **name**
  The new name of the node

**DESCRIPTION**

This function renames a node in the workcell hierarchy.

**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>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 node does not exist.</td>
</tr>
<tr>
<td>CX_NAME_CONFLICT</td>
<td>The new node name conflicts with other node under same parent.</td>
</tr>
<tr>
<td>CX_CANT_RENAME_WORLD</td>
<td>The node is world, which cannot be renamed.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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 picks a peg from the table with tcf. It then changes the name of the peg from peg_on_the_table to peg_at_finger_tip.

```c
CxServer Server;
CxNodeId table, tcf, peg;
.
.
/* add a new node named peg_on_the_table to table */
CxAddNewNode (Server, "peg_on_the_table ", table);

/* get peg’s CxNodeId */
CxGetNamedNodeId (Server, "peg_on_the_table ", &peg);

/* set rigid_flag so peg is not rigidly attached to its parent */
CxSetNodeRigid (Server, peg, CX_FALSE);
```
/* move robot such that the finger tcf is at the peg */

/* pick up the peg with the finger tcf */
CxPickUp(Server, peg, tcf);

/* change peg's name */
CxRenameNode(Server, peg, "peg_at_finger_tip");

SEE ALSO

CxAddNewNode, CxDeleteNode
CxRestoreDefaultState

Restores the default workcell state

SYNOPSIS

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

ARGUMENTS

- `Server` The Server ID

DESCRIPTION

This function restores the default workcell state from the default state file. A state file describes the hierarchy of the workcell, the pose, view (on or off), and rendering of all nodes in the workcell. Currently, the default workcell state file `default.st` is located in the directory where the CIMServer executable is started. When a workcell is loaded, the state is saved in a file called “default.st”.

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>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_FILE_OPEN_ERROR</td>
<td>The named state file does not exist.</td>
</tr>
<tr>
<td>CX_HIERARCHY_RESTORE_FAILED</td>
<td>The workcell hierarchy cannot be restored.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following code saves the initial state of the workcell, does some work in the workcell, and then restores the saved state.

```c
CxServer Server;

/* save the current state as default */
CxSaveDefaultState (Server);

/* workcell modifications such as pick/place operations and/or rigid body updating */

/* restore the default state */
CxRestoreDefaultState (Server);
```
SEE ALSO

CxSaveState, CxRestoreState, CxSaveDefaultState
CxRestoreState

Restores the workcell state from a file

SYNOPSIS

```c
#include <code/robpac.h>
long CxRestoreState (CxServer Server, char *state_name)
```

ARGUMENTS

- `Server`: The server ID
- `state_name`: The state file name

DESCRIPTION

This function restores a workcell state from a file. A state file describes the hierarchy of the workcell, the pose, view (on or off), and rendering of all nodes in the workcell. If no path is specified with `state_name`, the function will search the current working directory for the named state file.

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>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_FILE_OPEN_ERROR</td>
<td>The named state file does not exist.</td>
</tr>
<tr>
<td>CX_HIERARCHY_RESTORE_FAILED</td>
<td>The workcell hierarchy cannot be restored.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following example saves the initial state of the workcell, modifies the workcell, and then restores the saved state. CxSaveState is used to save the state that will be restored with CxRestoreState.

```c
CxServer Server;

/* save current state as statel.st */
CxSaveState (Server, "statel.st");

/* workcell modifications such as pick/place operations and/or rigid body updating */

/* restore the previously saved state */
CxRestoreState (Server, "statel.st");
```
SEE ALSO

CxSaveState, CxRestoreDefaultState, CxSaveDefaultState
CxRigidBody1

Update the position of an object based upon the position of a single fiducial

SYNOPSIS
#include <code/RigidBody.h>

void CxRigidBody1 (CxMatrix t_c_m, double x, double y,
                    double theta, CxVector ang, CxVector vec);

ARGUMENTS
  t_c_m  A matrix representing the Transformation from the Camera to the Mechanism base.
  X      The X position of the fiducial (X vision result).
  Y      The Y position of the fiducial (Y vision result).
  Theta  The θ angle or orientation of the fiducial, in degrees (θ vision result).
  Ang    A vector representing the computed angle transformation.
  Vec    A vector representing the computed position transformation.

DESCRIPTION
Rigid body updating uses the fact that the relationship between many parts of an object are fixed; if you know the relationship between the parts of the object, and you know the position of one part of the object, then you can determine where all of the other parts are located. For example, if I know the design a measurements of a table, and I know where one of the tables legs is located, I know where all of the other table legs are also. In addition, if I move one of the table legs, I know that the other table legs will also move, because they are all rigidly attached to the table top.

In robotic, rigid body updating is commonly used to update the position of a PCB (printed circuit board), based upon the position of one or a few fiducials. A fiducial is a special mark or pattern on the PCB that is easily located using a vision system, and whose position is know with respect other points on the PCB. Suppose that you want to place several components on the PCB, and that the placement positions are defined relative to the corner of the PCB.

You can model this using the CODE node hierarchy with the PCB modeled as a node with several child nodes, one representing each component place location. For CxRigidBody1, you would also define a fiducial node that is a child of the PCB node. Suppose that you also have a robot that has a camera and tool mounted to it, and that the camera and tool have been calibrated, so that the CODE system knows where they are with respect to each other and to the base of the robot (refer to CIM Cal for details on how to perform this calibration).

To update the position of the PCB, you would move the camera over the fiducial. Have the camera take a picture, and compute the X, Y, and θ of the fiducial. You would also, call CxFindrm to get the current position of the camera with respect to the base of the robot. Call CxRigidBody1 with this information and it will return a pose (angle and position vectors) that can be used to update the position of the PCB. After updating the PCB node’s position, you can command the tool to move directly to the component locations.

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

EXAMPLE
CxNodeId pcb;   // initialize to PCB node
CxNodeId fiducial;  // initialize to a fiducial node that is a
                    // child of PCB
CxNodeId camera;   // initialize to camera node, which is
                    // attached to robot
CxNodeId mech_node; // initialize to the base node of the robot
CxMechanism mech;  // initialize to the robot
CxServer server;  // server connection
CxVector ang;   // An angle vector that will be computed by
// CxRigidBody1
CxVector vec;   // A position vector that will be computed by
// CxRigidBody2
double x, y, theta; // The vision result
CxMatrix t_c_m;   // A transformation matrix representing the
// position of the camera with respect to the
// mechanism when the picture was taken

// Move camera to nominal position of fiducial.
// If PCB node is not reset to nominal position before
// this call, then this will move the camera to the
// last position where the fiducial was found.
CxMoveToNode (mech, fiducial, camera_tcf);

// Take a picture and process vision here, theta in degrees
// (Actual vision calls will depend upon your vision system)
TakePicture ();
If (FindFiducial (&x, &y, &theta)) {
    printf ("Error: could not find fiducial");
    return;
}

// get current cam_tcf CxPose
CxFindrm(server, camera, mech_node, t_c_m);

// This call (we pass in t_c_m, x, y and theta
// and get back ang and vec)
CxRigidBody1 (t_c_m, x, y, theta, ang, vec);

// Do rigid body updating.
// This updates the position of the PCB, such that the fiducial
// is at the position specified by ang and vec.
CxSetPose (server, pcb, mech_node, fiducial, "ZYX", ang, vec);

SEE ALSO
CxRigidBody2, CIMCal
**CxRigidBody2**

Updates the position of an object based upon the position of two fiducials

**SYNOPSIS**

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

void CxRigidBody2 (CxMatrix t_c_m_1, CxMatrix t_c_m_2,
                   CxMatrix t_ref2_ref1,
                   double x1, double y1, double x2, double y2,
                   CxVector ang, CxVector vec);
```

**ARGUMENTS**

- `t_c_m_1` A matrix representing the Transformation from the Camera to the Mechanism base, when viewing fiducial number 1.
- `t_c_m_2` A matrix representing the Transformation from the Camera to the Mechanism base, when viewing fiducial number 2.
- `t_ref2_ref1` A matrix representing the Transformation from the second fiducial to the first fiducial.
- `x1` The X position of fiducial 1 (X vision result).
- `y1` The Y position of the fiducial 1 (Y vision result).
- `x2` The X position of fiducial 2 (X vision result).
- `y2` The Y position of the fiducial 2 (Y vision result).
- `Ang` A vector representing the computed angle transformation.
- `Vec` A vector representing the computed position transformation.

**DESCRIPTION**

Rigid body updating uses the fact that the relationship between many parts of an object are fixed; if you know the relationship between the parts of the object, and you know the position of one part of the object, then you can determine where all of the other parts are located. For example, if I know the design a measurements of a table, and I know where one of the tables legs is located, I know where all of the other table legs are also. In addition, if I move one of the table legs, I know that the other table legs will also move, because they are all rigidly attached to the table top.

In robotic, rigid body updating is commonly used to update the position of a PCB (printed circuit board), based upon the position of one or a few fiducials. A fiducial is a special mark or pattern on the PCB that is easily located using a vision system, and whose position is know with respect other points on the PCB. Suppose that you want to place several components on the PCB, and that the placement positions are defined relative to the corner of the PCB.

You can model this using the CODE node hierarchy with the PCB modeled as a node with several child nodes, one representing each component place location. For CxRigidBody2, you would also define two fiducial nodes that are children of the PCB node. These fiducial nodes are often at opposite ends of the PCB. Suppose that you also have a robot that has a camera and tool mounted to it, and that the camera and tool have been calibrated, so that the CODE system knows where they are with respect to each other and to the base of the robot (refer to CIMCal for details on how to perform this calibration).

To update the position of the PCB, you would move the camera over one fiducial. Have the camera take a picture, and compute the X and Y position of the fiducial. You would also call CxFindrm to get the current position of the camera with respect to the base of the robot. You would then move the camera over the second fiducial and have the vision system compute the X, Y for this second fiducial, and call CxFindrm again to get the new position of the camera. Finally, call CxFindrm to get the relationship of the two fiducials. Call CxRigidBody2 with this information and it will return a pose (angle and position vectors) that can be used to update the position of the PCB. After updating the PCB node's position, you can command the tool to move directly to the component locations.
This function differs from CxRigidBody1 in that it uses two fiducials, instead of one, and the vision system does not need to find the orientation of the fiducial, only the position. Using two fiducials is generally more accurate that when using a single fiducial.

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

EXAMPLE
CxNodeId pcb;   // initialize to PCB node
CxNodeId fiducial1; // initialize to a fiducial node that is a
// child of PCB
CxNodeId fiducial2; // initialize to a different fiducial node
// that is a child of PCB
CxNodeId camera; // initialize to camera node, which is
// attached to robot
CxNodeId mech_node; // initialize to the base node of the robot
CxMechanism mech; // initialize to the robot
CxServer server; // server connection
CxVector ang;   // An angle vector that will be computed by
// CxRigidBody1
CxVector vec;   // A position vector that will be computed by
// CxRigidBody2
double x1, y1; // The vision result for the first fiducial
double x2, y2; // The vision result for the second fiducial
CxMatrix t_c_m_1; // A transformation matrix representing the
// position of the camera with respect to the
// mechanism when the picture of fiducial 1
// was taken
CxMatrix t_c_m_2; // A transformation matrix representing the
// position of the camera with respect to the
// mechanism when the picture of fiducial 2
// was taken
CxMatrix t_ref2_ref1; // A transformation matrix representing the
// relationship between fiducial 2 and
// fiducial 1.

// Move camera to nominal position of fiducial 1.
// If PCB node is not reset to nominal position before
// this call, then this will move the camera to the
// last position where the fiducial was found.
CxMoveToNode (mech, fiducial1, camera_tcf);

// Take a picture and process vision here
// (Actual vision calls will depend upon your vision system)
TakePicture ();
If (FindFiducial (&x1, &y1)) {
    printf ("Error: could not find fiducial");
    return;
}

// get current cam_tcf CxPose
CxFindrm(server, camera, mech_node, t_c_m_1);

// Move above the second fiducial
CxMoveToNode (mech, fiducial2, camera_tcf);

// Take a picture and process vision here
// (Actual vision calls will depend upon your vision system)
TakePicture ();
If (FindFiducial (&x1, &y1)) {
printf ("Error: could not find fiducial");
return;
}

// get current cam_tcf CxPose
CxFindrm(server, camera, mech_node, t_c_m_2);

// Find the relationship between fid2 and fid1
CxFindrm (g_server, fiducial2, fiducial1, t_ref2_ref1);

CxRigidBody2 (t_c_m_1, t_c_m_2, t_ref2_ref1, x1, y1, x2, y2,
ang, vec);

// Do rigid body updating.
// This updates the position of the PCB, such that fiducial 1
// is at the position specified by ang and vec.
CxSetPose (server, pcb, mech_node, fiducial, "ZYX", ang, vec);

SEE ALSO
CxRigidBody1, CIMCal
CxSaveDefaultState

Saves the current workcell state to the default state file

SYNOPSIS

```
#include <code/robpac.h>
long CxSaveDefaultState (CxServer Server)
```

ARGUMENTS

Server    The server ID

DESCRIPTION

This function saves the current workcell state to the default state file, default.st. A state file describes the hierarchy of the workcell, the pose, view (on or off), and rendering of all nodes in the workcell only. To save all the information of the workcell, use CxSaveWorkcell. The default state file default.st will be created in the directory where the CIMServer is started.

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>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_FILE_OPEN_ERROR</td>
<td>The new state file cannot be saved.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following example uses CxSaveDefaultState to save the current state for future use.

```

CxServer Server;
/* initialization */
/* save the current state as the default state */
CxSaveDefaultState (Server);
/* workcell modifications such as pick/place operations and/or rigid body updating */
/* restore the default state as necessary */
CxRestoreDefaultState (Server);
```

SEE ALSO

CxRestoreState, CxSaveState, CxRestoreDefaultState
CxSaveState

Saves the workcell state to a file

SYNOPSIS

```
#include <code/robpac.h>
long CxSaveState (CxServer Server, char *state_name)
```

ARGUMENTS

- **Server**  
  The server ID
- **state_name**  
  The state file name

DESCRIPTION

This function saves a workcell state to a file. A state file describes the hierarchy of the workcell, the pose, view (on or off), and rendering of all nodes in the workcell only. To save all the information of the workcell, use `CxSaveWorkcell`. If no path is specified, the file will be saved in the directory from which the process was started.

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>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_FILE_OPEN_ERROR</td>
<td>The new state file cannot be saved.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following example saves current workcell state in `catch_IC.st` file. The file will be created in the directory from which this process is launched.

```c

CxServer Server;

/* save the current state */
if (CxSaveState (Server, "catch_IC.st")!= CX_ERROR)
{
    printf ("Current mechanism state is saved in catch_IC.st \n file in the current directory.\n");
}


```

SEE ALSO

CxRestoreState, CxSaveDefaultState, CxRestoreDefaultState
**CxSaveWorkcell**

Saves the workcell to a file

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxSaveWorkcell (CxServer Server, char *filename, long ff, CxNodeId
start, long file_branch)
```

**ARGUMENTS**

- **Server**
  - The Server ID
- **filename**
  - The name of the file to be written. For a filename without a path, the CIMServer
    will output the named file to current working directory of the CODE application
    process. For a filename which includes a path, the system will store the workcell
    file to the specified directory.
- **ff**
  - The file format: **CX_ROBLINE_FORMAT**, **CX_IGES_FORMAT**.
- **start**
  - The starting node in the hierarchy tree where filing starts
- **file_branch**
  - The flag which indicates whether to file the start node and its children (**CX_TRUE**),
    or just the start node (**CX_FALSE**)

**DESCRIPTION**

This function saves a workcell to a file. The workcell can be saved in **CX_ROBLINE_FORMAT**, or
**CX_IGES_FORMAT**.

**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>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_FILE_OPEN_ERROR</td>
<td>The given workcell file cannot be saved.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>The given starting node does not exist.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following example saves current workcell into file **test.w** with **CX_ROBLINE_FORMAT**. The
workcell file **test.w** will be in the directory where this process was started.

```c
CxServer Server;
CxNodeId world;
.
.
if(CxSaveWorkcell (Server,"test.w",CX_ROBLINE_FORMAT,world,CX_TRUE)
   ==CX_ERROR) {
    printf("CX_ERROR in saving workcell as test.w\n");
```

} else {
    printf("Workcell saved to test.w in ROBLINE format.\n");
}

If a full path is included, the workcell file will be filed as specified. The following statement saves the currently loaded workcell to "test.w" in C:\Tom\cells.

CxSaveWorkcell (Server,"C:\Tom\cells\test.w", CX_ROBLINE_FORMAT, world,CX_TRUE);

SEE ALSO

CxLoadWorkcell,CxClearWorkcell,CxFileDefaultWorkcell
CxSearchUp

Searches up the tree for an ancestor with the desired attributes

SYNOPSIS

```
#include <code/robpac.h>
long CxSearchUp(CxServer Server, CxNodeId start_node, unsigned long want_mask, long ignore_self, CxNodeId *node_found)
```

ARGUMENTS

- **Server**
  The server ID
- **start_node**
  The node from which to start searching up
- **want_mask**
  The attribute mask of the desired node
- **ignore_self**
  Whether to ignore the start node itself (CX_TRUE or CX_FALSE)
- **node_found**
  The first node that is found

DESCRIPTION

This function searches up the tree from the `start_node`. The function will return the first node it encounters with the attribute specified by `want_mask`. The possible attributes that can be used for `want_mask` are:

- CX_FRAME_MASK
- CX_ROBOT_MASK
- CX_JOINT_MASK
- CX_GEOM_MASK
- CX_COMM_MASK
- CX_TOOL_MASK
- CX_REGION_MASK
- CX_FEATURE_MASK
- CX_SENSOR_MASK
- CX_TCF_MASK
- CX_TEACH_MASK
- CX_USER_MASK
- CX_TRAIL_MASK
- CX_ALL_MASK

Additionally, you may search for a group of attributes by bitwise ORing multiple masks together. For example, if you want to search for nodes with mechanism or joint attributes, you would enter the `want_mask` as follows: `CX_ROBOT_MASK | CX_JOINT_MASK`. This would return nodes with the mechanism attribute, the joint attribute or both.

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>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 start node does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The start node has an invalid node ID.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The start node has been cut out.</td>
</tr>
<tr>
<td>CX_SEARCH_UP_FAILED</td>
<td>No ancestor node has the specified attributes.</td>
</tr>
</tbody>
</table>
EXAMPLE
The following example shows how CxSearchUp can be used to find the mechanism node when any node in the mechanism model is selected as a start node.

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

void main(void)
{
    CxServer Server;
    char *server_name;
    CxNodeId node_selected;
    long branch;
    char node_name[CX_MAXNAME];

    if( CxGetDefaultServer( &server_name ) == CX_ERROR ) {
        printf("No default server found\n");
        exit( -1 );
    }

    Server = CxOpenServer(server_name, CX_SMEM, 0);

    printf("Select any node on the mechanism from the graphics \n" "window.\n");
    printf("Enter return when done.\n");
    fflush( stdin );
    getchar();
    CxGetSelectedNode(Server, &node_selected, &branch);

    if( CxSearchUp(Server, node_selected, CX_ROBOT_MASK, CX_FALSE,
        &node_found) == CX_ERROR ) {
        printf("CxSearchUp failed \n");
    } else {
        CxGetNodeName (Server, node_found, node_name);
        printf("Robot node name: %s\n", node_name);
    }
    .
    .
    CxRobpacExit();
}

SEE ALSO
CxGetNodeInfo, CxTraverse
**CxSetIgesTol**

Sets the IGES conversion tolerance

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxSetIgesTol (CxServer Server, double tol)
```

**ARGUMENTS**

- **Server**  
The Server ID
- **tol**  
The IGES conversion tolerance (≥ 0.05)

**DESCRIPTION**

This function sets the IGES conversion tolerance. The tolerance must be greater than or equal to 0.05.

IGES (Initial Graphics Exchange Specification) uses an arc center point, a starting point, and a terminate point to represent a circle or circular arc. The CIMServer database, on the other hand, uses edges and polygons to approximate a circle and a cylinder. The number of edges used to create the element is referred to as the resolution. Tolerance refers to the difference between the radius of the true circle, and the minimum radius of the polygonalized circle. The radius and tolerance, or the resolution can be used to determine the edges and polygons.

The higher the resolution or the lower the tolerance, the closer the approximation is to a true circle. Resolution and tolerance are dependent. Once a resolution is given for a circle with a known radius, tolerance is calculated, and vice versa. IGES tolerance is the tolerance used to convert an IGES circular entity into CIMServer elements.

The following figure depicts the relationship between resolution, tolerance, and radius.

![Diagram showing relationship between resolution, tolerance, and radius](image)

**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>An error in sending the message.</td>
</tr>
<tr>
<td>CX_MESSAGE_RECEIVE_FAILED</td>
<td>An error in receiving the message.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

CxGetIgesTol
CxSetNodeRigid

Sets the node’s rigid attachment status

SYNOPSIS

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

ARGUMENTS

- **Server**  The Server ID
- **node**  The node ID
- **rigid_flg**  The flag which determines whether the node is rigidly attached to its parent (CX_TRUE or CX_FALSE)

DESCRIPTION

This function sets the node’s rigid attachment status. When a node is rigidly attached to its parent, the node by itself, cannot be picked up. When a node is rigidly attached to its parent, both the node and its parent will move when either is picked up. In other words, one cannot modify the position of a node that is rigidly attached to its parent without changing the parent as well. When a node is created, the default rigid status is set to CX_TRUE (rigidly attached).

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>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_INVALID_NODE_ID</td>
<td>The 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_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following example checks the rigid status for the node **peg** before picking it up with **gripper**. If node **peg** is rigidly attached to its parent, The status is set to CX_FALSE (not rigidly attached).

```

CxServer Server;
CxNodeId peg, gripper;
long rigidly_attached;

/* check if peg is rigidly attached to its parent */
CxGetNodeRigid(Server, peg, &rigidly_attached);
if (rigidly_attached)
```
{ 
    CxSetNodeRigid(Server, peg, CX_FALSE);
}

/* pick up the peg */
CxPickUp (Server, peg, gripper);

SEE ALSO
    CxGetNodeRigid
CxSetNodeSelected

Selects a node (or branch) in the workcell

SYNOPSIS

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

ARGUMENTS

- **Server**: The Server ID
- **node**: The node ID
- **branch**: The flag which sets whether a node and its children will be selected (CX_TRUE) or whether just the named node will be selected (CX_FALSE)
- **highlight**: A flag that lets the server know whether or not to highlight the node in the Graphics Window. If CX_TRUE, the node will be highlighted in the Graphics Window.

DESCRIPTION

This function selects the node in the workcell. If the `branch` flag is set, the selected node and all its child nodes will be selected. If `highlight` is set, all selected nodes will be highlighted on the graphics screen.

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>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 given 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_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following example asks the user to input the name of the node to be selected, and whether or not the user wants to select the branch. The function then selects the node (and branch) of interest.

```c
CxServer Server;
char name[CX_MAXNAME];
CxNodeId node;

long branch;
long hilite;
char yesno;
```
printf("Which node do you want to select: ");
fflush(stdin);
scanf(" %s", name);

printf("Do you want to select branch as well (Y/N)? ");
fflush(stdin);
scanf(" %c", &yesno);
if (yesno == 'y' || yesno == 'Y') {
    branch = CX_TRUE;
} else {
    branch = CX_FALSE;
}

printf("Do you want to highlight branch as well (Y/N)? ");
fflush(stdin);
scanf(" %c", &yesno);
if (yesno == 'y' || yesno == 'Y') {
    hilite = CX_TRUE;
} else {
    hilite = CX_FALSE;
}

if (CxGetNamedNodeId (Server, name, &node) != CX_ERROR)
{
    CxSetNodeSelected (Server, node, branch, hilite);
}

SEE ALSO

CxGetSelectedNode
**CxSetUnit**

Sets the current linear unit conversion factor and the unit used for angular quantities

**SYNOPSIS**

```c
#include <code/robpac.h>
void CxSetUnit (double linear_unit_factor, long angle_unit_flag)
```

**ARGUMENTS**

- **linear_unit_factor** The unit conversion factor for linear quantities
- **angle_unit_flag** Unit used for angular quantities. Its value can be **CX_USE_DEGREE**, which is the default setting, or **CX_USE_RADIAN**.

**DESCRIPTION**

This function sets the current linear conversion factor and the unit used for angular quantities. Linear quantities used in various API functions are multiplied by the `linear_unit_factor` when passed from a CODE application process to the CIMServer, and divided by the `linear_unit_factor` when passed in the other direction. The default value of `linear_unit_factor` is 1.0. This implies that linear values are sent without any change between the CODE application process and the CIMServer.

The unit for angular quantities can be either **CX_USE_DEGREE** or **CX_USE_RADIAN**. The CIMServer always uses radians for angular quantities, so, if a CODE application process sets its angular unit to **CX_USE_DEGREE**, angular values will be converted to radians before being sent to the Server. Also, values returned by the Server will be converted from radians to degrees before being returned to the process. If the process has its angular unit set to **CX_USE_RADIANS**, the value will be passed between the client and the Server without any conversion. The default value of `angle_unit_flag` is **CX_USE_DEGREE**.

**NOTE:** This API only affects distances used for modeling functions, but does not have any effect on the mechanism’s speed.

**RETURN VALUES**

This function does not return any values.

**EXAMPLE**

The following example sets the `linear_unit_factor` to 2 and selects radian to be the unit for angular quantities.

```c
CxSetUnit (2.0, CX_USE_RADIAN);
```

**SEE ALSO**

- `CxGetUnit`
CxTraverse
Traverses the tree looking for nodes with specific attributes

SYNOPSIS
#include <code/robpac.h>
CxNodeInfo * CxTraverse (CxServer Server, CxNodeInfo *start, CxNodeInfo *node, unsigned long want_mask, unsigned long block_mask, CxNodeInfo *node_blk)

ARGUMENTS
Server The server ID
start The node from which to start traversing
node The current node (node which is found)
want_mask The attribute mask of the desired node
block_mask The search will not traverse down the branch of any node with this attribute
node_blk The search will not traverse down the branch of this node

DESCRIPTION
This function traverses all of the nodes in the tree whose first node is start. This function will return when it finds the first node that matches the want_mask. However, if used as shown in the example, it will find all the nodes that match the want_mask. The function will not traverse any branch whose first node has the attribute specified by block_mask. It will similarly not traverse any branch whose first node is specified by node_blk. The possible attributes that can be used for want_mask or block_mask are:

<table>
<thead>
<tr>
<th>CX_FRAME_MASK</th>
<th>CX_ROBOT_MASK</th>
<th>CX_JOINT_MASK</th>
<th>CX_GEOM_MASK</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_COMM_MASK</td>
<td>CX_TOOL_MASK</td>
<td>CX_REGION_MASK</td>
<td>CX_FEATURE_MASK</td>
</tr>
<tr>
<td>CX SENSOR_MASK</td>
<td>CX_TCF_MASK</td>
<td>CX_TEACH_MASK</td>
<td>CX_USER_MASK</td>
</tr>
<tr>
<td>CX TRAIL_MASK</td>
<td>CX ALL_MASK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Additionally, you may search for a group of attributes by bitwise ORing multiple masks together. For example, if you want to search for nodes with mechanism or joint attributes, you would enter the want_mask as follows: CX_ROBOT_MASK | CX_JOINT_MASK. This will return all nodes with the mechanism attribute, the joint attribute, or both. You may do the same thing for the block_mask.

RETURN VALUES
This function returns a pointer to a valid CxNodeInfo data structure if successful; otherwise, a CX_NULL pointer 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>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 specified node does not exist.</td>
</tr>
<tr>
<td>INVALID_NODE_ID</td>
<td>The node number does not match node index number.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The given node has been cut out.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following example shows how `CxTraverse` can be used to list all of the mechanisms and joints in a workcell.

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

void main(void)
{
    CxServer Server;
    CxNodeId world;
    CxNodeInfo *curNode, start, tmpNode;

    Server = CxOpenServer("ServerName", CX_SMEM, 0);
    CxGetNamedNodeId (Server, "world", &world);
    CxGetNodeInfo (Server, world, &start);
    tmpNode.number = 0; tmpNode.node_index = 0;
    curNode = &tmpNode;

    printf("Nodes with mechanism or joint attributes:\n");
    while(curNode = CxTraverse (Server, &start, curNode,
        CX_ROBOT_MASK|CX_JOINT_MASK, 0, CX_NULL)){
        printf(" %s\n", curNode->name);
    }

    CxRobpacExit();
}
```

**SEE ALSO**

`CxGetNodeInfo`, `CxSearchUp`
Position and Orientation
CxFindrm

Finds the relative pose matrix between two workcell nodes

SYNOPSIS

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

ARGUMENTS

- **Server**: The Server ID
- **node**: The node ID
- **ref_node**: The node ID for reference node
- **rel_mat**: Relative pose matrix of `node` with respect to `ref_node`

DESCRIPTION

This function determines the relative pose matrix between two workcell 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>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 given node or <code>ref_node</code> 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_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following example gets the ID’s of node1 and node2. It then finds the position matrix for node2 with respect to node1.

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

void main(void)
{
    CxServer    Server;
    CxNodeId   node1, node2;
    CxMatrix    trans_matrix;

    /* get node ID’s */
    CxGetNamedNodeId (Server, "Node1", node1);
    CxGetNamedNodeId (Server, "Node2", node2);
```
/* Get transformation matrix of Node2 with respect to Node1 */
findrm (Server, node2, node1, trans_matrix);


CxRobpacExit();

}

SEE ALSO

CxFindJntIndepRm, CxGetPose, CxSetPose, CxGetRelPose, CxSetRelPose
**CxFindJntIndepRm**

Finds the joint-independent relative pose matrix between two workcell nodes

**SYNOPSIS**

```
#include <code/robpac.h>
long CxFindJntIndepRm (CxServer Server, CxNodeId node, CxNodeId ref_node, CxMatrix rel_mat)
```

**ARGUMENTS**

- `Server` The Server ID
- `node` The node ID
- `ref_node` The node ID for a reference node
- `rel_mat` The relative pose matrix of a node with respect to `ref_node`

**DESCRIPTION**

This function determines the joint-independent relative pose matrix between two workcell nodes. The API functions `CxFindJntIndepRm` and `Cxfindrm` are exactly the same if none of the nodes in the hierarchy between `node` and `ref_node` has the JOINT attribute. However, if there are nodes having the JOINT attribute, `CxFindJntIndepRm` will return the pose as if the joints were at their zero state, while `Cxfindrm` will return the pose with the joint values included in the calculation.

**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>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 given node or <code>ref_node</code> 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_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following code will first get the node IDs of the nodes. This robot has the following hierarchy: `robot->joint->tcf`. The function prints out the absolute and present transformation matrix of the tcf node with respect to robot node. It then moves the joint and prints out these values again. Observe that the current transformation matrix changes with respect to the robot node depending on the position of the joint node but the absolute transformation matrix remains the same.

```c
#include <code/robpac.h>
void main(void)
{
    CxServer    Server;
    CxMechanism mech;
    CxNodeId    robot, joint, tcf;
```
Cxmatrix trans_matrix;

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

CxGetNamedNodeId (Server, "robot", &robot);
CxGetNamedNodeId (Server, "joint", &joint);
CxGetNamedNodeId (Server, "tcf", &tcf);

/* Get current position of tcf with respect to the robot */
CxFindrm (Server, tcf, robot, trans_matrix);
printf ("Current position matrix is: \n");
CxPrintm (trans_matrix);

/* Get absolute position of tcf with respect to the robot */
CxFindJntIndepRm (Server, tcf, robot, trans_matrix);
printf ("Absolute position matrix is: \n");
CxPrintm (trans_matrix);
mech = CxOpenMechanism (Server, robot, CX_CONTROL);
CxMoveSingleAxis (mech, 0, 30.0);

/* Get current position of tcf with respect to the robot */
CxFindrm (Server, tcf, robot, trans_matrix);
printf ("Current position matrix is: \n");
CxPrintm (trans_matrix);

CxFindJntIndepRm (Server, tcf, robot, trans_matrix);
printf ("Absolute position matrix is: \n");
CxPrintm (trans_matrix);

CxRobpacExit();
}

SEE ALSO
CxFindrm, CxGetJntIndepPose, CxGetPose, CxSetPose, CxGetRelPose, CxSetRelPose
CxGetJntIndepPose

Gets the joint independent pose relating a node to a reference node

SYNOPSIS

#include <code/robpac.h>
long CxGetJntIndepPose(CxServer Server, CxNodeId node, CxNodeId ref,
char axes[4], CxVector angles, CxVector vec)

ARGUMENTS

Server  The Server ID
node    The node ID
ref     The node relative to which the pose of the node is to be determined
axes    Returned the principal axes about which the frame is rotated, and the order of the rotation
         (i.e. ZYX, XYZ, etc.)
angles   Returns the relative rotation angles in the same order as the axes were returned. The
         returned angles are in degrees unless the unit is set to CX_USE_RADIAN. See
         CxSetUnit in the Nodes, Frames and Attributes section of the CODE API
vec      Returns the relative position vector

NOTE: Type CxVector is defined in <code/matx_defs.h> as follows:
     typedef double CxVector[3];

DESCRIPTION

This function gets the joint independent pose of a node (usually represented by a coordinate frame) relative to
another node. The API functions CxGetJntIndepPose and CxGetPose are exactly the same if none of
the nodes in the hierarchy between node and ref has the JOINT attribute. However, if there are nodes with
the JOINT attribute, CxGetJntIndepPose will return the pose representing where
node would be with respect to “ref” if all joints were at their zero state. CxGetPose will return the pose representing where
node is with respect to ref with the joint values included in the calculation.

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>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 given node (either node or ref) does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The given node (either node or ref) 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>
The following code gets the node IDs of the nodes. The robot has following hierarchy: robot->joint->tcf. The function prints out the absolute and present positions of the tcf node with respect to robot node. Then it moves the joint and prints out these values again. The current position and orientation values change with respect to robot node depending on the position of the joint node, but the absolute position and orientation values remain the same.

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

void main(void)
{
    CxServer    Server;
    CxMechanism mech;
    CxNodeId   robot, joint, tcf;
    CxVector    angles, pos;

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

    CxGetNamedNodeId( Server, "robot", &robot );
    CxGetNamedNodeId( Server, "joint", &joint );
    CxGetNamedNodeId( Server, "tcf", &tcf );

    /* Get current position of tcf with respect to the robot */
    CxGetPose( Server, tcf, robot, "XYZ", angles, pos );
    printf(" Current position is : %f %f %f \n", pos[0], pos[1], pos[2] );
    printf(" Current orientation is : %f %f %f \n\n", angles[0], angles[1], angles[2] );

    /* Get absolute position of tcf with respect to the robot */
    CxGetJntIndepPose( Server, tcf, robot, "XYZ", angles, pos );
    printf(" Absolute position is : %f %f %f \n", pos[0], pos[1], pos[2] );
    printf(" Absolute orientation is : %f %f %f \n\n", angles[0], angles[1], angles[2] );

    mech = CxOpenMechanism( Server, robot, CX_CONTROL );
    CxMoveSingleAxis( mech, 0, 30.0 );

    /* Get current position of tcf with respect to the robot */
    CxGetPose( Server, tcf, robot, "XYZ", angles, pos );
    printf(" Current position is : %f %f %f \n", pos[0], pos[1], pos[2] );
    printf(" Current orientation is : %f %f %f \n\n", angles[0], angles[1], angles[2] );

    /* Get current position of tcf with respect to the robot */
    CxGetJntIndepPose( Server, tcf, robot, "XYZ", angles, pos );
    printf(" Absolute position is : %f %f %f \n", pos[0], pos[1], pos[2] );
    printf(" Absolute orientation is : %f %f %f \n\n", angles[0], angles[1], angles[2] );
```

CxRobpacExit();
}

SEE ALSO
CxSetJntIndepPose, CxGetPose, CxGetRelPose
CxGetPose

Gets the pose relating a node to a reference node

SYNOPSIS

```c
#include <code/robpac.h>
long CxGetPose(CxServer Server, CxNodeId node, CxNodeId ref, char axes[4], CxVector angles, CxVector vec)
```

ARGUMENTS

- **Server**: The Server ID
- **node**: The node ID
- **ref**: The node relative to which the pose of node (the node ID) is to be determined
- **axes**: Returns the principal axes about which the frame is rotated, and the order of the rotation (e.g. Z,X,Y; X,Y,Z; etc.)
- **angles**: Returns the relative rotation angles in the same order as the axes were returned. The returned angles are in degrees unless the unit is set to CX_USE_RADIAN. See example and CxSetUnit in the Nodes, Frames and Attributes section of the CODE API Programmer's Reference Manual- Volume 2 for more information.
- **vec**: Returns the relative position vector

**NOTE:** Type `CxVector` is defined in `<code/matx_defs.h>` as follows:

```c
typedef double CxVector[3];
```

DESCRIPTION

This function gets the pose of a node (usually represented by a coordinate frame) relative to another node.

WARNINGS

The principal axes are the X, Y, and Z axes of the node frame; thus, a rotation about one axis changes the orientation of the other two.

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>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 given node (either node or ref) does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The given node (either node or ref) 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 code will get the pose of node part relative to another node table. It returns the rotation axes (axes, in the order the rotations take place), the rotation angles (ang, in degrees, about those axes in the same order), and the relative position vector (vec).

```c
CxServer Server;
char axes[4];
CxVector ang, vec;
CxNodeId part, world;

CxGetPose( Server, part, world, axes, ang, vec);

/* If you want CxGetPose to return angles in radians, simply do following: */

CxSetUnit(1.0, CX_USE_RADIAN);
CxGetPose( Server, part, world, axes, ang, vec);

Note: The unit set with CxSetUnit affects all successive API functions, except functions in Matrix library. If necessary, the units can always be set back to the preferred unit.

SEE ALSO

CxGetRelPose, CxGetJntIndepPose, CxSetRelPose, CxSetPose, CxXyzmat
CxGetRelPose

Gets pose relating a node to its parent node

SYNOPSIS
#include <code/robpac.h>
long CxGetRelPose(CxServer Server, CxNodeId node, char axes[4],
CxVector angles, CxVector vec )

ARGUMENTS
Server Server ID
node Node ID
axes Returns the principal axes about which the frame is rotated, and the order of the rotation
(e.g. ZYX; XYZ; etc.)
angles Returns the relative rotation angles in the same order as the axes were returned. The
returned angles are in degrees unless the unit is set to CX_USE_RADIAN. See the
example and CxSetUnit in the Nodes, Frames and Attributes section of the CODE
vec Returns the relative position vector

NOTE: Type CxVector is defined in <code/matx_defs.h> as follows:
typedef double CxVector[3];

DESCRIPTION
The pose of a node (usually represented by a coordinate frame) relative to its parent node can be found with
CxGetRelPose.

WARNINGS
The principal axes are the X, Y, and Z axes of the node frame. Thus, a rotation about one axis changes the
orientation of the other two.

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>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 given 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_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>
EXAMPLE

The following example code will get the pose of node part relative to its parent node table. It returns the rotation axes (axes, in the order the rotations take place), the rotation angles (ang, in degrees, about those axes in the same order), and the relative position vector (vec).

```c
CxServer Server;
CxNodeId part;
char axes[4];
CxVector ang, vec;

CxGetRelPose( Server, part, axes, ang, vec);

/* If you want CxGetRelPose to return angles in radians, simply do following: */

CxSetUnit(1.0, CX_USE_RADIAN);
CxGetRelPose( Server, part, axes, ang, vec);

Note: The unit set with CxSetUnit affects all successive API functions, except functions in the Matrix library. If necessary, the units can always be set back to the preferred unit.

SEE ALSO

CxSetRelPose, CxGetPose, CxSetPose, CxXyzmat
CxGetToolPose

Gets the pose relating a tool control frame (TCF) to the robot base frame

SYNOPSIS

```c
#include <code/robpac.h>
long CxGetToolPose( CxMechanism mech, CxNodeId tcf, char axes[4],
CxVector angles, CxVector vec )
```

ARGUMENTS

- **mech**: The mechanism ID
- **tcf**: The tool control frame node ID
- **axes**: The order of rotation for angles (e.g. ZYX)
- **angles**: The relative rotation angles. The returned angles are in degrees unless the unit is set to CX_USE_RADIAN. See the example and CxSetUnit in the Nodes, Frames and Attributes section of the CODE API Programmers Reference Manual- Volume 2 for more information.
- **vec**: The relative position vector

**NOTE:** Type CxVector is defined in `<code/matx_defs.h>` as follows:

```c
typedef double CxVector[3];
```

DESCRIPTION

This function is used to get the pose relating the specified tool control frame to the robot base 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>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_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>The given node does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_NODE_ID</td>
<td>The node number does not match the node index number.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The given node has been cut out.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following code uses CxGetToolPose to get the current pose of the tool control frame of a camera (tcf) being used for calibration. The function CxXyzmat (from the Matrix library) is called to take the pose information returned from CxGetToolPose and create a standard 4x4 transformation (or pose) matrix describing the pose of the tcf relative to the robot's base frame.
CxMechanism mech;
CxNodeId tcf;
char axes[4];
CxVector ang, vec;
CxMatrix mat;

/* set defaults unit */
CxSetUnit(1.0, CX_USE_RADIAN);

/* get current pose of the vision camera TCF */
CxGetToolPose( mech, tcf, axes, ang, vec);
CxXyzmat( axes, ang, vec, mat);

Note: This example uses radian as the default unit since matrix library functions such as CxXyzmat only take angles in radians.

SEE ALSO
CxGetRelPose
CxSetDeltaPose

Changes a node’s relative pose with respect to its current pose by some delta pose

SYNOPSIS

```
#include <code/robpac.h>
long CxSetDeltaPose(CxServer Server, CxNodeId node, char axes[4],
                    CxVector angles, CxVector vec)
```

ARGUMENTS

- **Server**: The Server ID
- **node**: The node ID
- **axes**: The order of rotation for angles (e.g. ZYX)
- **angles**: The relative rotation angles. The angles are in degrees unless the unit is set to CX_USE_RADIANS. See CxSetUnit in the Nodes, Frames and Attributes section of the CODE API Programmer’s Reference Manual- Volume 2 for more information.
- **vec**: The relative position vector

**NOTE:** Type CxVector is defined in <code/matx_defs.h> as follows:
```
typedef double CxVector[3];
```

DESCRIPTION

This function is used to change a node’s relative pose with respect to itself by some delta pose.

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>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 given 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_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following example creates a path in space. It then rotates the reference frame of that path through 360 degrees, effectively rotating all the segments belonging to the path.
```
#include <code/robpac.h>
void main(void)
{
    CxServer   Server;
    CxNodeId  path;
    double   angle, ang_inc;
    CxVector    delta_angle, delta_position;
```
/* Add a new circular segment to the path */
CxAddCurveSeg( Server, path, "", "pt1", "ZXY",
0.0, 0.0, 0.0, 0.0, 0.0, 0.0);
CxAddCurveSeg( Server, path, "pt1", "pt2", "ZXY",
0.0, 30.0, 0.0, 250.0, 0.0, 0.0);

/* Rotate the complete path about Y axis by 360 degrees */
ang_inc = 10.0;
delta_angle[0] = ang_inc;
delta_angle[1] = 0.0;
delta_angle[2] = 0.0;

delta_position[0] = 5.0;
delta_position[1] = 0.0;
delta_position[2] = 0.0;

for (angle=0.0; angle< 361.0; angle+=ang_inc) {
    CxSetDeltaPose( Server, path, "YZX", delta_angle, delta_position);
}

SEE ALSO
CxGetRelPose
**CxSetJntIndepPose**

Sets a node’s joint-independent pose with respect to a reference node

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxSetJntIndepPose(CxServer Server, CxNodeId node, CxNodeId ref,
                       char axes[4], CxVector angles, CxVector vec )
```

**ARGUMENTS**

- **Server**
  - The Server ID
- **node**
  - The node ID
- **ref**
  - The node relative to which the pose of node is to be set
- **axes**
  - The principal axes about which the frame is rotated, and the order of the rotation (e.g. ZYX, XYZ, etc.)
- **angles**
  - Returns the relative rotation angles in the order specified by axes. The angles are in degrees unless the unit is set to CX_USE_RADIAN. See the example and CxSetUnit in the Nodes, Frames and Attributes section of the CODE API Programmers Reference Manual- Volume 2 for more information.
- **vec**
  - The relative position vector

**NOTE:** Type CxVector is defined in `<code/matx_defs.h>` as follows:

```
typedef double CxVector[3];
```

**DESCRIPTION**

This function sets the joint independent pose of a node relative to another node. The function CxSetJntIndepPose will set the pose of the node with respect to ref, ignoring offsets caused by nodes with the JOINT attribute at a non-zero state.

**WARNINGS**

The principal axes are the X, Y, and Z axes of the node frame. Thus, a rotation about one axis changes the orientation of the other two.

**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 CxGetErrNumber 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>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 given node (either node or ref) 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_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>
SEE ALSO

CxGetJntIndepPose, CxSetPose, CxGetRelPose, CxSetRelPose
**CxSetPose**

Does a rigid body correction for a node

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxSetPose(CxServer Server, CxNodeId node, CxNodeId m_frame,
               CxNodeId meas_ele, char axes[4], CxVector angles, CxVector vec)
```

**ARGUMENTS**

- **Server**
  The Server ID
- **node**
  The node ID for the node to be updated
- **m_frame**
  The measurement frame (sensor)
- **meas_ele**
  The element being measured (fiducial)
- **axes**
  The order of rotation for angles (e.g. ZYX)
- **angles**
  The relative rotation angles. The angles are in degrees unless the unit is set to CX_USE_RADIAN. See CxSetUnit in the *Nodes, Frames and Attributes* section of the CODE API Programmers Reference Manual- Volume 2 for more information.
- **vec**
  The relative position vector

**NOTE:** Type CxVector is defined in <code/matx_defs.h> as follows:

```c
typedef double CxVector[3];
```

**DESCRIPTION**

This function is used to do a rigid body update for a node. The parameters axes, angles, and vec represent the pose of meas_ele (fiducial) measured with respect to m_frame (sensor). This measured pose information is then used to update the pose of the node in the workcell file. The parameters node and meas_ele must be rigidly attached to each other. This API function allows node and meas_ele to be the same node (see the following example).

**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>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 given 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_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>
EXAMPLE

The following example uses a sensor (normally a vision camera or similar device) to measure the pose of a certain fiducial in the workcell with the `measure_pose` function (a user-defined measurement function). The functions `CxSetJointSpeed`, `CxMoveRelNode`, and `CxMoveToNode` are used to move the sensor to the fiducial.

After the program calls `measure_pose`, it calls `CxSetPose` to change the ideal workcell pose value of the fiducial (a node in the workcell model) to reflect the pose of the actual, physical fiducial. Finally, in this particular case, `CxSaveState` is called to save the current workcell state as the default.

```c
CxMechanism mech;
CxServer Server;
CxNodeId fiducial;
Sensor *sensor;
double mspeed = 0.7, aspeed = 0.1;
CxVector ang, vec;
char axes[4];

/* move sensor near fiducial */
CxSetJointSpeed( mech, mspeed );
CxMoveRelNode( mech, fiducial, sensor->snsr_id, "ZYX", 0., 0., 0., 0., 0., 15. );

/* move sensor to the fiducial feature */
CxSetJointSpeed(mech, aspeed);
CxMoveToNode( mech, fiducial, sensor->snsr_id );

/* measure fiducial pose relative to the sensor frame
measure_pose is a custom measurement function */
measure_pose( mech, fiducial, sensor, 1, axes, ang, vec );
/* update workcell model with measured pose */
CxSetPose( Server, fiducial, sensor->snsr_id, fiducial, axes, ang, vec );

/* save current workcell state */
CxSaveDefaultState(server);

SEE ALSO

CxUpdateTcf, CxGetPose, CxGetRelPose, CxSetRelPose, CxUpdatePose
```
**CxSetRelPose**

Changes a node’s relative pose with respect to its parent

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxSetRelPose(CxServer Server, CxNodeId node, char axes[4],
        CxVector angles, CxVector vec)
```

**ARGUMENTS**

- **Server**
  The Server ID
- **node**
  The node ID
- **axes**
  The order of rotation for angles (e.g. XYZ)
- **angles**
  The relative rotation angles. The angles are in degrees unless the unit is set to `CX_USE_RADIAN`. See `CxSetUnit` for more information.
- **vec**
  The relative position vector

**NOTE:** Type `CxVector` is defined in `<code/matx_defs.h>` as follows:

```c
typedef double CxVector[3];
```

**DESCRIPTION**

This function is used to set a node’s pose relative to its parent.

**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>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 given 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_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following example makes a trapezoid which sits on top of an existing box. The trapezoid is added as a child of the existing box `base_box`. The trapezoid will have upper length and width equal to half the corresponding dimensions on the box, and lower length and width the same as the box. The height of the trapezoid will be twice that of the box. The `CxGetBox` function will be used to determine the dimensions of the trapezoid, while the functions `CxAddNewNode`, `CxAddGeometry`, and `CxMakeTrap` are called (in that order) to make the trapezoid.

The `CxSetRelPose` function is used in this code to ensure that the trapezoid is placed in a reasonable pose relative to the box. In this case, the trapezoid has the same orientation (all angles of rotation equal to 0) as the box, and is located just on top (relative Z distance equal to `z`, the height of the box).
CxServer Server;
CxNodeId box, trap;
double x, y, z, l1, w1, l2, w2, h;
long i;
CxVector ang, vec;

/* get base_box dimensions */
if( CxGetNamedNodeId(Server, "base_box", &box) == CX_ERROR ) {
    fprintf(stderr,"CX_ERROR: base_box does not exist.\n");
    CxRobpacExit();
}
CxGetBox( Server, box, &x, &y, &z );

/* add node trap as a child of base_box */
CxAddNewNode( Server, "trap", box );
CxGetNamedNodeId( Server, "trap", &trap );

/* set relative pose for trap with respect to base_box frame */
for(i=0; i<3; i++) {
    ang[i] = vec[i] = 0.0;
}
vec[2] = z;
CxSetRelPose( Server, trap, "ZYX", ang, vec);

/* set trap dimensions */
l1 = x / 2.0;
w1 = y / 2.0;
l2 = x;
w2 = y;
h = 2. * z;

/* add geometry attribute to trap node */
CxAddGeometry( Server, trap );

/* make the desired trapezoid */
CxMakeTrap( Server, trap, l1, w1, l2, w2, h);

SEE ALSO
CxGetRelPose, CxGetPose, CxSetPose
**CxUpdatePose**  
Updates the pose of a node with respect to another node in the workcell

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxUpdatePose(CxServer Server, CxNodeId node, CxNodeId ref_node, 
        char axes[4], CxVector angles, CxVector vec)
```

**ARGUMENTS**

- `Server` The Server ID
- `node` The node ID for the node to be updated
- `ref_node` The node ID for the reference node
- `axes` The order of rotation for angles (e.g. ZYX)
- `angles` The relative rotation angles. The angles are in degrees unless the unit is set to `CX_USE_RADIAN`. See CxSetUnit in the Nodes, Frames and Attributes section of the CODE API Programmers Reference Manual- Volume 2 for more information.
- `vec` The relative position vector

**NOTE:** Type `CxVector` is defined in `<code/code/matx_defs.h>` as follows:

```c
typedef double CxVector[3];
```

**DESCRIPTION**

This function updates the pose matrix of the `node` with respect to its parent node such that its pose with respect to the `ref_node`. The pose is specified by `axes`, `angles`, and `vec`.

This function can be used to do a rigid body updating for a node. It can also be used for TCF updating. The `ref_node` can be the immediate parent of the `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><code>CX_MESSAGE_SEND_FAILED</code></td>
<td>An error in sending the message.</td>
</tr>
<tr>
<td><code>CX_MESSAGE_RECEIVE_FAILED</code></td>
<td>An error in receiving the message.</td>
</tr>
<tr>
<td><code>CX_NODE_NOT_FOUND</code></td>
<td>The given node does not exist.</td>
</tr>
<tr>
<td><code>CX_NODE_IS_CUT_OUT</code></td>
<td>The 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>
<tr>
<td><code>NON_XYZ_AXIS</code></td>
<td>The axis specified is not valid.</td>
</tr>
<tr>
<td><code>PARENT_NOT_FOUND</code></td>
<td>The node does not have a parent. and cannot be updated.</td>
</tr>
</tbody>
</table>
The following example uses a sensor (probably a vision camera or similar sensor device) to measure the pose of a fiducial (node) with respect to the sensor TCF in the workcell with the measure_pose function (a user-defined measurement function). The functions CxSetJointSpeed, CxMoveRelNode, and CxMoveToNode are used to move the sensor to the feature.

After the program calls measure_pose, it calls CxUpdatePose to change the ideal workcell pose of the fiducial with respect to its parent to reflect the pose of the actual, physical fiducial. Finally, in this particular case, CxSaveState is called to save the current workcell state as the default.

```c
CxMechanism mech;
CxServer Server;
CxNodeId fiducial;
Sensor *sensor;
double mspeed = 0.7, aspeed = 0.1;
CxVector ang, vec;
char axes[4];

/* move sensor near fiducial with fast speed */
CxSetJointSpeed( mech, mspeed );
CxMoveRelNode( mech, fiducial, sensor->snsr_id, "ZYX", 0., 0.,
0., 0., 0., 15. );

/* move sensor to the fiducial with slower speed */
CxSetJointSpeed( mech, aspeed );
CxMoveToNode( mech, fiducial, sensor->snsr_id );

/* measure fiducial pose relative to the sensor frame
measure_pose is a custom measurement function */
measure_pose( mech, fiducial, sensor, 1, axes, ang, vec );

/* update cell model with measured pose */
CxUpdatePose( Server, fiducial, sensor->snsr_id, axes, ang, vec);

/* save current workcell state */
CxSaveDefaultState( Server );
```

SEE ALSO

CxUpdateTcf, CxGetPose, CxGetRelPose, CxSetRelPose, CxSetPose
Mechanism Parameters
**CxFileDHParm**

Writes a serial mechanism’s DH parameters to a file

**SYNOPSIS**

```
#include <code/robpac.h>
long CxFileDHParm(CxMechanism mech, CxNodeId mech_id, char *file_name)
```

**ARGUMENTS**

- **mech** The mechanism ID
- **mech_id** The mechanism node ID
- **file_name** The path and name of the file where the DH parameters will be filed

**DESCRIPTION**

This function calls the automatic Denavit-Hartenberg (DH) classification procedure, and files the generated DH parameters in a user-named file. If `file_name` includes a path, i.e. the name of a directory, the file is stored in that location. If it does not contain a path, the file is stored in the current working directory. If a path is specified, the DH parameters generated are filed in that directory. These procedures will only work if the mechanism has serial joints, or if the mechanism is a simple two-branch NC mechanism.

**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>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 given mechanism node does not exist.</td>
</tr>
<tr>
<td>CX_NODE.IS.CUT.OUT</td>
<td>The node is cut out and cannot be modified.</td>
</tr>
<tr>
<td>CX_NON_NC.OR_SERIAL_MECHANISM</td>
<td>The non-serial mechanism, or simple two branch NC mechanism.</td>
</tr>
<tr>
<td>CX_FILE.OPEN_ERROR</td>
<td>Cannot save the specified file.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
</tbody>
</table>
**CxGetAccelTimesMin**

Gets the minimum allowed rise and fall acceleration times

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxGetAccelTimesMin(CxMechanism mech, CxNodeId mech_id,
                        double *rise_time_min, double * fall_time_min)
```

**ARGUMENTS**

- `mech` The mechanism ID
- `mech_id` The mechanism node ID
- `rise_time_min` The minimum time to accomplish a speed increase
- `fall_time_min` The minimum time to accomplish a speed decrease

**DESCRIPTION**

The function allows the user to get the minimum allowable time to accomplish the rise and fall accelerations when time is used to govern the speed change.

**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>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 given mechanism node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The node is cut out and cannot be modified.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the ROBOT attribute.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following example gets the minimum acceleration and deceleration times and modifies them if appropriate.

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

void main(void)
{
    CxServer    Server;
    CxMechanism mech;
    CxNodeId    mechanism, tcf, t1, t2, t3;
    double     rise_time, fall_time;
    long       increase_accel_limit, decrease_accel_limit;

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

```
 CXGetNamedNodeId( Server, "mechanism", &mechanism );
 CXGetNamedNodeId( Server, "tcf", &tcf );
 CXGetNamedNodeId( Server, "t1", &t1 );
.
.
 mech = CxOpenMechanism( Server, mechanism, CX_CONTROL );
 CXSetBlendPolicy( mech, CX_MOVE_TO);
 CXSetInterpType( mech, CX_LINEAR_INTERP );
 CXSetAccelType( mech, CX_CONST_RAMP_TIME );

 CXGetAccelTimesMin( mech, mechanism, &rise_time, &fall_time );
 /* Set velocity profile parameters */
 if (increase_accel_limit)
   CXSetAccelTimesMin( mech, mechanism, rise_time*0.9,
                        fall_time*0.9 );
 else if (decrease_accel_limit)
   CXSetAccelTimesMin( mech, mechanism, rise_time*1.1,
                        fall_time*1.1 );
 /* Move mechanism to accomplish a task */
 CXMoveToNode( mech, t1, tcf );
 CXMoveToNode( mech, t2, tcf );
 CXMoveToNode( mech, t3, tcf );
.
.
 CxWaitForEndOfMotion( mech );
 CXRobpacExit();
} 

WARNINGS

Since allowable minimum rise and fall acceleration times are mechanism- and design-dependent, the user
must be extremely cautious when entering these values if they will be used to actually control a mechanism.
Excessive accelerations may cause the mechanism’s inertial loads to damage the drive transmissions and
other mechanical components on the mechanism.

SEE ALSO

 CXSetAccelTimesMin
CxGetActualScrewSpeed

Gets the current screw speed

SYNOPSIS

```
#include <code/robpac.h>
long CxGetActualScrewSpeed(CxMechanism mech, CxNodeId mech_id, double *speed)
```

ARGUMENTS

- `mech`: The mechanism ID
- `mech_id`: The mechanism node ID
- `speed`: The current actual screw speed (deg/sec or rad/sec, depending on units. The default unit is degrees. See `CxSetUnit` in the Nodes, Frames and Attributes section of the CODE API Programmers Reference Manual- Volume 2 for more information).

DESCRIPTION

This function gets the current (or actual) screw speed of a mechanism. This function differs from `CxGetScrewSpeed` (which returns the screw speed setting) in that it returns a value which is dependent on the move type, target, orientation, blending status, and whether any joints have exceeded their rate limits. The actual rotational speed about a screw vector is used to rotate the tool frame to the target frame. This function is valid only if the moving mechanism is not undergoing joint interpolation. If the mechanism is not moving, 0.0 will be returned.

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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The node is cut out and cannot be modified.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the ROBOT attribute.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following example gets the maximum screw speed and sets it to a new value as appropriate. It continuously monitors its value during the motion. When making a move from one point to another, two types of speeds are involved: a linear speed to reach from one point to another, and an angular speed to make the two frames align with each other (assuming motion is a full pose). The angular speed is set by these 'screw' functions. During the motion, both the angular and the linear speeds are taken into account. Whichever takes the maximum time governs the motion.

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

void main(void)
```
{  
    CxServer    Server;
    CxMechanism mech;
    CxNodeId   MyMech, tcf, t1, t2, t3;
    double max_speed, tool_speed, screw_speed;
    long increase_speed_limit, decrease_speed_limit;
    Server = CxOpenServer( "Testing", CX_SMEM, 0 );
    CxGetNamedNodeId( Server, "MyMech", &MyMech );
    CxGetNamedNodeId( Server, "tcf", &tcf );
    CxGetNamedNodeId( Server, "t1", &t1 );

    mech = CxOpenMechanism( Server, MyMech, CX_CONTROL );
    CxSetBlendPolicy( mech, CX_MOVE_TO);
    CxSetInterpType( mech, CX_LINEAR_INTERP );
    CxGetMaxScrewSpeed( mech, MyMech, &max_speed );

    /* Set motion parameters */
    if (increase_speed_limit)
        CxSetMaxScrewSpeed( mech, MyMech, max_speed*1.1 );
    else if (decrease_speed_limit)
        CxSetMaxScrewSpeed( mech, MyMech, max_speed*0.9 );

    /* Move MyMech to accomplish a task */
    CxMoveToNode( mech, t1, tcf );
    CxMoveToNode( mech, t2, tcf );
    CxMoveToNode( mech, t3, tcf );

    while (1)  
    {  
        CxGetActualScrewSpeed( mech, MyMech, &screw_speed );
        printf(" Current screw speed : %lf \n", screw_speed );
        .
    }
    CxWaitForEndOfMotion( mech );
    CxRopacExit();
}

SEE ALSO

CxGetScrewSpeed, CxSetScrewSpeed
**CxGetActualToolSpeed**

Gets the current tool speed for a moving mechanism

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxGetActualToolSpeed(CxMechanism mech, CxNodeId mech_id, double *speed)
```

**ARGUMENTS**

- `mech`: The mechanism ID
- `mech_id`: The mechanism node ID
- `speed`: The current tool speed (in/sec, mm/sec, etc.)

**DESCRIPTION**

This function gets the current linear tool speed of a mechanism. It is not to be confused with `CxGetTipSpeed`, which returns the tool speed setting. This function is valid only if the moving mechanism is not undergoing joint-interpolated motion. If the mechanism is not moving, 0.0 will be returned.

**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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The node is cut out and cannot be modified.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the ROBOT attribute.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following example gets the maximum tool speed and sets it to a new value as appropriate. It continuously monitors its value during the motion. When making a move from one point to another, two speed types are involved: a linear speed to reach from one point to another, and an angular speed to make the two frames align with each other (assuming motion is a full pose). The linear speed is set by these 'tool' functions. During the motion, both the angular and the linear speeds are taken into account and whichever takes the maximum time governs the motion.

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

void main(void)
{
    CxServer    Server;
    CxMechanism mech;
    CxNodeId   MyMech, tcf, t1, t2, t3;
    double     max_speed, tool_speed, screw_speed;
```
long increase_speed_limit, decrease_speed_limit;

Server = CxOpenServer( "Testing", CX_SMEM, 0 );
CxGetNamedNodeId( Server, "MyMech", &MyMech );
CxGetNamedNodeId( Server, "tcf", &tcf );
CxGetNamedNodeId( Server, "t1", &t1 );

mech = CxOpenMechanism( Server, MyMech, CX_CONTROL );
CxSetBlendPolicy( mech, CX_MOVE_TO);
CxSetInterpType( mech, CX_LINEAR_INTERP );

CxGetMaxToolSpeed( mech, MyMech, &max_speed );
/* Set motion parameters */
if (increase_speed_limit)
    CxSetMaxToolSpeed( mech, MyMech, max_speed*1.1 )
else if (decrease_speed_limit)
    CxSetMaxToolSpeed( mech, MyMech, max_speed*0.9 );

/* Move MyMech to accomplish a task */
CxMoveToNode( mech, t1, tcf );
CxMoveToNode( mech, t2, tcf );
CxMoveToNode( mech, t3, tcf );

while (1)
{
    CxGetActualToolSpeed( mech, MyMech, &tool_speed );
    printf(" Current tip speed : %lf \n", tool_speed );
    .
}

Cx WaitForEndOfMotion( mech );
CxRobpacExit();

SEE ALSO

CxGetTipSpeed, CxSetTipSpeed
**CxGetDefaultSoln**

Gets the default inverse kinematics solution for the mechanism

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxGetDefaultSoln (CxMechanism mech, CxNodeId mech_id, long *soln)
```

**ARGUMENTS**

- `mech` The mechanism ID
- `mech_id` The mechanism node ID
- `soln` The default inverse kinematic solution for the mechanism

**DESCRIPTION**

This function gets the default inverse kinematic solution to be used in doing an interpolated move. When `CX_ALL_SOLN` (-1) is used as the parameter for `soln`, all solutions are searched and the closest one is used. Any other number uses a specific solution.

**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>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 specified node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The node is cut out and cannot be modified.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the ROBOT attribute.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following example sets the default solution for the mechanism to be of type `CX_ALL_SOLN`. This forces the mechanism to use the closest solution.

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

void main(void)
{
    CxMechanism mech;
    CxNodeId   MyMech;
    long      soln_used;

    Server = CxOpenServer( "Testing", CX_SMEM, 0);
    CxGetNamedNodeId( Server, "MyMech", &MyMech );
    mech = CxOpenMechanism( Server, MyMech, CX_CONTROL );
    CxSetDefaultSoln( mech, MyMech, CX_ALL_SOLN );
}
See also

CxSetDefaultSoln
**CxGetFclassNumber**

Gets a mechanism’s forward kinematics classification (fclass) number

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxGetFclassNumber(CxMechanism mech, CxNodeId mech_id, long *class)
```

**ARGUMENTS**

- `mech`: The mechanism ID
- `mech_id`: The mechanism node ID
- `class`: The mechanism's forward kinematics classification number

**DESCRIPTION**

This function gets the forward kinematics classification (fclass) number for a given mechanism. The fclass number is used to call a user-supplied routine that determines dependent joint values and joint limits for mechanisms having more complex structures where joint limits may vary as a function of mechanism configuration.

**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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The node is cut out and cannot be modified.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the ROBOT attribute.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

CxSetFclassNumber, CxGetFwrdkinType, CxSetFwrdkinType
CxGetFwrdkinType

Gets a mechanism's forward kinematics solution type

SYNOPSIS

#include <code/robpac.h>
long CxGetFwrdkinType(CxMechanism mech, CxNodeId mech_id, long *type)

ARGUMENTS

mech The mechanism ID
mech_id The mechanism node ID
type The forward kinematics solution type

DESCRIPTION

This function gets the forward kinematics type for a given mechanism. The allowable values are either
CX_INDEPEND (only independent joints present), CX_FUNCTION (some joints are functionally dependent
on others) or CX_LIN_DEP (some joints are linearly dependent on others).

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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The node is cut out and cannot be modified.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the ROBOT attribute.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following example gets the forward kinematics solution type set for the mechanism and prints it out.
#include <code/robpac.h>

void main(void)
{
    CxServer Server;
    CxMechanism mech;
    CxNodeId MyMech;
    long soln_type;

    Server = CxOpenServer( "Testing", CX_SMEM, 0 );
    CxGetNamedNodeId( Server, "MyMech", &MyMech );
    mech = CxOpenMechanism( Server, MyMech, CX_CONTROL );

    CxGetFwrdkinType( mech, MyMech, &soln_type );
printf(" MyMech’s forward kinematics solution type is: ");
if ( soln_type == CX_INDEP )
    printf(" independent
 " );
else if ( soln_type == CX_FUNCTION )
    printf(" functionally dependent
 " );
else if ( soln_type == CX_LIN_DEP )
    printf(" linearly dependent
 " );
else
    printf(" unknown
 " );
CxRobpacExit();
}

SEE ALSO

CxSetFwrdkinType, CxGetFclassNumber, CxSetFclassNumber
CxGetIclassNumber

Gets a mechanism’s inverse kinematics classification (iclass) number

SYNOPSIS

#include <code/robpac.h>
long CxGetIclassNumber(CxMechanism mech, CxNodeId mech_id, long *class)

ARGUMENTS

mech     The mechanism ID
mech_id   The mechanism node ID
class    The mechanism's inverse kinematics number classification

DESCRIPTION

This function gets the mechanism’s inverse kinematics classification (iclass) number for a given mechanism. The iclass number is used by CIMServer to access either the default inverse kinematics routine or a user-supplied custom inverse kinematics routine for a particular mechanism(s), depending upon inverse kinematics type specified by CxSetInvkinType. If the inverse kinematics solution type is set to CX_AUTO_INVKIN, then CxGetIclassNumber will attempt to match the mechanism to a DH solution.

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>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 given mechanism node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The node is cut out and cannot be modified.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the ROBOT attribute.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
</tbody>
</table>

SEE ALSO

CxSetIclassNumber, CxGetInvkinType, CxSetInvkinType
CxGetInvkinType

Gets a mechanism’s inverse kinematics solution type

SYNOPSIS

```c
#include <code/robpac.h>
long CxGetInvkinType(CxMechanism mech, CxNodeId mech_id, long *type)
```

ARGUMENTS

- `mech`: The mechanism ID
- `mech_id`: The mechanism node ID
- `type`: The inverse kinematics solution type (see list in DESCRIPTION)

DESCRIPTION

This function gets the inverse kinematics solution type for a given mechanism. The following are the allowable types:

- `CX_AUTO_INVKIN`: CX_AUTO_DH procedures
- `CX_CUSTOM_INVKIN`: User-supplied function
- `CX_NUMERICAL_INVKIN`: Numerical inverse kinematics procedures
- `CX_NO_INVKIN`: No inverse kinematics

The CIMServer’s numerical inverse kinematics routine should be used only for simulation purposes.

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>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 given mechanism node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The node is cut out and cannot be modified.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the ROBOT attribute.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
</tbody>
</table>

SEE ALSO

CxSetInvkinType, CxGetIclassNumber, CxSetIclassNumber
CxGetJclassNumber

Gets a mechanism’s joint rate classification (jclass) number

SYNOPSIS

```
#include <code/robpac.h>
long CxGetJclassNumber(CxMechanism mech, CxNodeId mech_id, long *jclass)
```

ARGUMENTS

- **mech**: The mechanism ID
- **mech_id**: The mechanism node ID
- **jclass**: The mechanism joint rate classification number

DESCRIPTION

This function gets the current joint rate classification (jclass) number for a given mechanism. The jclass number is used by the CIMServer to call a user-supplied routine to calculate joint rates, given trajectory following of motion segments at specified speeds.

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>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 given mechanism node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The node is cut out and cannot be modified.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the ROBOT attribute.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
</tbody>
</table>

SEE ALSO

CxSetJclassNumber
### CxGetJnt2actMap

Gets the joint space to actuator space mapping number

#### SYNOPSIS

```c
#include <code/robpac.h>
long CxGetJnt2actMap (CxMechanism mech, long *jnt2act_map)
```

#### ARGUMENTS

- **mech**: Mechanism ID. A `CxMechanism` id is returned from a call to `CxOpenMechanism`.
- **jnt2act_map**: Joint space to actuator space mapping number

#### DESCRIPTION

This function gets the current number setting of the joint space to actuator space mapping number of the given mechanism. The default is a one-to-one mapping of joint values and speeds from one space to the other.

#### 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 Code</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_INVALID_MECHANISM</td>
<td>Invalid mechanism ID, or mechanism does not exist.</td>
</tr>
</tbody>
</table>

#### WARNINGS

If `jnt2act_map` is set to a non-zero value (zero being the CIMServer default), the user has to make sure that the appropriate code segments are added in the files `user.motion.AJ` and `user.mmotion.JA` which are found in the `$ROBTOP/lib/cimetrix/custom` directory. See the `Customizing CODE` manual for more information.
CxGetMaxScrewSpeed

Gets a mechanism’s maximum screw speed setting

SYNOPSIS

#include <code/robpac.h>
long CxGetMaxScrewSpeed(CxMechanism mech, CxNodeId mech_id, double *max_speed)

ARGUMENTS

mech The mechanism ID
mech_id The mechanism node ID
max_speed The maximum screw speed (deg/sec or rad/sec, depending on units. The default unit is degrees. See CxSetUnit in the Nodes, Frames and Attributes section of the CODE API Programmers Reference Manual- Volume 2 for more information).

DESCRIPTION

This function gets maximum screw speed setting for a given mechanism. The actual screw speed attained may be less than this setting.

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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The node is cut out and cannot be modified.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the ROBOT attribute.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following example gets the maximum screw speed and sets it to a new value as appropriate. It continuously monitors its value during the motion. When making a move from one point to another, two types of speeds are involved; a linear speed to reach from one point to another, and an angular speed to make the two frames align with each other (assuming motion is a full pose). The angular speed is set by these ‘screw’ functions. During the motion, both the angular and the linear speeds are taken into account and whichever takes the maximum time governs the motion.

#include <code/robpac.h>
void main( void )
{
    CxServer Server;
    CxMechanism mech;
    CxNodeId MyMech, tcf, t1, t2, t3;
    double max_speed, tool_speed, screw_speed;
long increase_speed_limit, decrease_speed_limit;

Server = CxOpenServer( "Testing", CX_SMEM, 0 );
CxGetNamedNodeId( Server, "MyMech", &MyMech );
CxGetNamedNodeId( Server, "tcf", &tcf );
CxGetNamedNodeId( Server, "t1", &t1 );

mech = CxOpenMechanism( Server, MyMech, CX_CONTROL );
CxSetBlendPolicy( mech, CX_MOVE_TO);
CxSetInterpType( mech, CX_LINEAR_INTERP );

CxGetMaxScrewSpeed( mech, MyMech, &max_speed );
/* Set motion parameters */
if (increase_speed_limit)
    CxSetMaxScrewSpeed( mech, MyMech, max_speed*1.1 );
else if (decrease_speed_limit)
    CxSetMaxScrewSpeed( mech, MyMech, max_speed*0.9 );
/* Move MyMech to accomplish a task */
CxMoveToNode( mech, t1, tcf );
CxMoveToNode( mech, t2, tcf );
CxMoveToNode( mech, t3, tcf );

while (1)
{
    CxGetActualScrewSpeed( mech, MyMech, &screw_speed );
    printf(" Current screw speed : %f \n", screw_speed );
    
}

CxWaitForEndOfMotion( mech );
CxRobpacExit();

SEE ALSO

CxSetMaxScrewSpeed, CxGetScrewSpeed, CxSetScrewSpeed, CxGetActualScrewSpeed
CxGetMaxToolSpeed

Gets the maximum tool speed setting

SYNOPSIS

#include <code/robpac.h>
long CxGetMaxToolSpeed(CxMechanism mech, CxNodeId mech_id, double *max_speed)

ARGUMENTS

mech  The mechanism ID
mech_id  The mechanism node ID
max_speed  The maximum tool speed (in/sec, mm/sec, etc., depending on units)

DESCRIPTION

This function gets the maximum tool speed setting for a given mechanism. The actual tool speed attained in curvilinear moves may be less than this value.

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>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 given mechanism node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The node is cut out and cannot be modified.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the ROBOT attribute.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following example gets the maximum tool speed and sets it to a new value as appropriate. It continuously monitors its value during the motion. When making a move from one point to another, two types of speeds are involved; a linear speed to reach from one point to another, and an angular speed to make the two frames align with each other (assuming that the tool motion type is CX_FULL_POSE). The linear speed is set by these 'tool' functions. During the motion, both the angular and the linear speeds are taken into account Whichever takes the maximum time governs the motion.

#include <code/robpac.h>

void main(void)
{
    CxServer    Server;
    CxMechanism mech;
    CxNodeId   mymech, tcf, t1, t2, t3;
    double     max_speed, tool_speed, screw_speed;

long      increase_speed_limit, decrease_speed_limit;

Server = CxOpenServer( "Testing", CX_SMEM, 0 );
CxGetNamedNodeId( Server, "mymech", &mymech );
CxGetNamedNodeId( Server, "tcf", &tcf );
CxGetNamedNodeId( Server, "t1", &t1 );
.
.
mech = CxOpenMechanism( Server, mymech, CX_CONTROL );
CxSetBlendPolicy( mech, CX_MOVE_TO);
CxSetInterpType( mech, CX_LINEAR_INTERP );

CxGetMaxToolSpeed( mech, mymech, &max_speed );
/* Set motion parameters */
if (increase_speed_limit)
    CxSetMaxToolSpeed( mech, mymech, max_speed*1.1 )
else if (decrease_speed_limit)
    CxSetMaxToolSpeed( mech, mymech, max_speed*0.9 );

/* Move mymech to accomplish a task */
CxMoveToNode( mech, t1, tcf );
CxMoveToNode( mech, t2, tcf );
CxMoveToNode( mech, t3, tcf );

while (1)
{
    CxGetActualToolSpeed( mech, mymech, &tool_speed );
    printf(" Current tip speed : %f \n", tool_speed );
    .
}

CxWaitForEndOfMotion( mech );
CxRobpacExit();

SEE ALSO
    CxSetMaxToolSpeed, CxGetToolSpeed, CxSetTipSpeed, CxGetActualToolSpeed
CxGetRclassNumber

Gets a mechanism’s redundant classification (rclass) number

SYNOPSIS

#include <code/robpac.h>
long CxGetRclassNumber(CxMechanism mech, CxNodeId node, long *class)

ARGUMENTS

mech The mechanism ID
mech_id The mechanism node ID
class The redundant classification (rclass) number

DESCRIPTION

This function gets the current redundant classification (rclass) number for a given mechanism. The redundant classification number is used to call a user-supplied routine which supplies weighting factors to the joints in the CIMServer numerical inverse kinematics routine. The numerical inverse kinematics routine should be used only for simulation purposes. Mechanisms with more than 5 independent joints may contribute differently to the overall motion of a mechanism. This function allows for balancing the way these joints contribute to the inverse kinematics calculations.

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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The node is cut out and cannot be modified.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the ROBOT attribute.</td>
</tr>
<tr>
<td>CXINVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
</tbody>
</table>

SEE ALSO

CxSetRclassNumber, CxGetInvkinType, CxSetInvkinType
CxGetScrewAccelMax

Gets the max screw accel and decel for Cartesian path following

SYNOPSIS

```c
#include <code/robpac.h>
long CxGetScrewAccelMax(CxMechanism mech, NodeId mech_id, double *accel_rise, double *accel_fall)
```

ARGUMENTS

- `mech` The mechanism ID
- `mech_id` The mechanism node ID
- `accel_rise` The maximum screw acceleration (deg/sec² or rad/sec², depending on units. The default unit is degrees. See `CxSetUnit` in the `Nodes, Frames and Attributes` section of the `CODE API Programmers Reference Manual- Volume 2` for more information.
- `accel_fall` The maximum screw deceleration (deg/sec² or rad/sec², depending on units. The default unit is degrees. See `CxSetUnit` in the `Nodes, Frames and Attributes` section of the `CODE API Programmers Reference Manual- Volume 2` for more information.

DESCRIPTION

This function allows the user to get the current maximum screw (rotational) acceleration and deceleration values, where the primary motion is rotation about a space axis.

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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The node is cut out and cannot be modified.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the ROBOT attribute.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following example gets the maximum trapezoidal acceleration and deceleration and modifies them as appropriate. When making a move from one point to another, two types of velocities are involved: a linear velocity required to reach from one point to another, and an angular velocity to make the two frames align with each other (assuming that the tool motion type is CX_FULL_POSE). When the acceleration type equals CX_CONST_RAMP_ACCEL, the angular acceleration and deceleration is set by these 'screw' functions.

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

void main(void)
```
Server = CxOpenServer( "Testing", CX_SMEM, 0 );
CxGetNamedNodeID( Server, "MyMech", &MyMech );
CxGetNamedNodeID( Server, "tcf", &tcf );
CxGetNamedNodeID( Server, "t1", &t1 );
.
.
mech = CxOpenMechanism( Server, MyMech, CX_CONTROL );
CxSetBlendPolicy( mech, CX_MOVE_TO);
CxSetInterpType( mech, CX_LINEAR_INTERP );
CxSetAccelType( mech, CX_CONST_RAMP_ACCEL );

CxGetScrewAccelMax( mech, MyMech, &accel, &decel );
/* Set motion parameters */
if (increase_accel_limit)
  CxSetScrewAccelMax( mech, MyMech, accel*1.1, decel*1.1 );
else if (decrease_accel_limit)
  CxSetScrewAccelMax( mech, MyMech, accel*0.9, decel*0.9 );

/* Move MyMech to accomplish a task */
CxMoveToNode( mech, t1, tcf );
CxMoveToNode( mech, t2, tcf );
CxMoveToNode( mech, t3, tcf );
.
.
CxWaitForEndOfMotion( mech );
CxRobpacExit();

**WARNINGS**

Since allowable maximums are mechanism- and design-dependent, the user must be extremely cautious when entering these values if they will be used to actually control a mechanism. Excessive accelerations may cause the mechanism’s inertial loads to damage the drive transmissions and other mechanical components on the mechanism.

**SEE ALSO**

CxSetScrewAccelMax, CxGetAccelType, CxSetAccelType
CxGetTclassNumber

Gets a mechanism’s trajectory classification (tclass) number

SYNOPSIS

#include <code/robpac.h>
long CxGetTclassNumber(CxMechanism mech, CxNodeId mech_id, long *class)

ARGUMENTS

mech  The mechanism ID
mech_id  The mechanism node ID
class  The trajectory classification (tclass) number

DESCRIPTION

This function gets the current trajectory classification (tclass) number for a given mechanism. The trajectory classification number is used to select a trajectory generation algorithm.

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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_NODE.IS_CUT_OUT</td>
<td>The node is cut out and cannot be modified.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the ROBOT attribute.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
</tbody>
</table>

SEE ALSO

CxSetTclassNumber
**CxGetTifFrame**

Gets a mechanism’s terminal interface frame (TIF)

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxGetTifFrame(CxMechanism mech, CxNodeId mech_id, CxNodeId *frame)
```

**ARGUMENTS**

- `mech` The mechanism ID
- `mech_id` The mechanism node ID
- `frame` The terminal interface frame node ID

**DESCRIPTION**

This function gets the terminal interface frame (TIF) for the given mechanism. Rather than using the last joint frame in the inverse kinematics calculations, manufacturers sometimes decide to use an interface frame which corresponds to the mounting interface for tools on mechanisms.

**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>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 given mechanism node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The node is cut out and cannot be modified.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the ROBOT attribute.</td>
</tr>
<tr>
<td>CX_TIF_NOT_FOUND</td>
<td>No TIF is defined for the mechanism.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following example gets the name of the TIF used by the mechanism and prints it out.

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

void main(void)
{
    CxServer    Server;
    CxMechanism mech;
    CxNodeId   MyMech, tif;
    char name [CX_MAXNAME];

    Server = CxOpenServer( "Testing", CX_SMEM, 0 );
    CxGetNamedNodeID( Server, "MyMech", &MyMech );
    mech = CxOpenMechanism( Server, MyMech, CX_CONTROL );

    /* Get the TIF name */
    CxGetTifName( mech, &name );
    printf("The TIF name is: %s\n", name);
}
```
CxGetTifFrame( mech, MyMech, &tif );
get_node_name ( Server, tif, name );
printf( "the tif frame for the given MyMech is %sn", name);
CxRobpacExit();

SEE ALSO

CxSetTifFrame
**CxGetTrapAccelMax**

Gets the max rise and fall accel settings for Cartesian path following

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxGetTrapAccelMax(CxMechanism mech, CxNodeId mech_id, double *accel_rise, double *accel_fall)
```

**ARGUMENTS**

- **mech**: The mechanism ID
- **mech_id**: The mechanism node ID
- **accel_rise**: The maximum positive acceleration value in linear units (mm/s², in/sec², etc.)
- **accel_fall**: The maximum positive deceleration value in linear units (mm/s², in/sec², etc.)

**DESCRIPTION**

The function allows the user to get the maximum rise and fall accelerations for trajectory following in Cartesian space.

**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>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 given mechanism node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The node is cut out and cannot be modified.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the ROBOT attribute.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following example gets the maximum trapezoidal acceleration and deceleration and modifies them as appropriate. When making a move from one point to another, two types of velocities are involved: a linear velocity required to reach from one point to another, and an angular velocity to make the two frames align with each other (assuming motion is a full pose). When the acceleration type equals CX_CONST_RAMP_ACCEL, the linear acceleration and deceleration are set by these ‘trap’ functions.

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

void main(void)
{
    CxServer    Server;
    CxMechanism mech;
```
CxNodeId   MyMech, tcf, t1, t2, t3;
double    accel, decel;
long      increase_accel_limit, decrease_accel_limit;

Server = CxOpenServer( "Testing", CX_SMEM, 0 );
CxGetNamedNodeId( Server, "MyMech", &MyMech );
CxGetNamedNodeId( Server, "tcf", &tcf );
CxGetNamedNodeId( Server, "t1", &t1 );
mech = CxOpenMechanism( Server, MyMech, CX_CONTROL );
CxSetBlendPolicy( mech, CX_MOVE_TO);
CxSetInterpType( mech, CX_LINEAR_INTERP );
CxSetAccelType( mech, CX_CONST_RAMP_ACCEL );

CxGetTrapAccelMax( mech, MyMech, &accel, &decel );
/* Set motion parameters */
if (increase_accel_limit)
    CxSetTrapAccelMax( mech, MyMech, accel*1.1, decel*1.1);
else if (decrease_accel_limit)
    CxSetTrapAccelMax( mech, MyMech, accel*0.9, decel*0.9 );

/* Move MyMech to accomplish a task */
CxMoveToNode( mech, t1, tcf );
CxMoveToNode( mech, t2, tcf );
CxMoveToNode( mech, t3, tcf );

CxWaitForEndOfMotion( mech );
CxRobpacExit();

WARNINGS
Since allowable maximums are mechanism- and design-dependent, the user must be extremely cautious when
entering these values if they will be used to actually control a mechanism. Excessive accelerations may cause
the mechanism's inertial loads to damage the drive transmissions and other mechanical components on the
mechanism.

SEE ALSO
CxSetTrapAccelMax, CxGetTrapAccelRamps, CxSetTrapAccelRamps, CxGetAccelType,
CxSetAccelType
CxSetAccelTimesMin
Sets a mechanism’s minimum rise and fall acceleration times

SYNOPSIS
#include <code/robpac.h>
long CxSetAccelTimesMin(CxMechanism mech, CxNodeId mech_id, 
double rise_time_min, double fall_time_min)

ARGUMENTS
mech The mechanism ID
mech_id The mechanism node ID
rise_time_min The minimum time to accomplish a speed increase (>= 0.005 sec)
fall_time_min The minimum time to accomplish a speed decrease (>= 0.005 sec)

DESCRIPTION
The function allows the user to set the minimum allowable time to accomplish the rise and fall accelerations when time is used to govern the speed change. Both numbers must be greater than or equal to 0.005 seconds.

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>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 given mechanism node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the ROBOT attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The node is cut out and cannot be modified.</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>The minimum time entered is less than 0.005 seconds.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>The mechanism is not opened for control.</td>
</tr>
</tbody>
</table>

EXAMPLE
The following example gets the minimum acceleration and deceleration times and modifies them as appropriate.

#include <code/robpac.h>

void main(void)
{
    CxServer    Server
Caution

Since allowable minimum rise and fall acceleration times are mechanism- and design-dependent, the user must be extremely cautious when entering these values if they will be used to actually control a mechanism. Excessive accelerations may cause the mechanism’s inertial loads to damage the drive transmissions and other mechanical components on the mechanism.

SEE ALSO

CxGetAccelTimesMin
CxSetDefaultSoln

Sets the inverse kinematic solution to be used

SYNOPSIS

```c
#include <code/robpac.h>
long CxSetDefaultSoln(CxMechanism mech, CxNodeId node, long soln)
```

ARGUMENTS

- mech: The mechanism ID
- node: The mechanism node ID
- soln: The inverse kinematic solution to be used

DESCRIPTION

Sets the inverse kinematic solution to be used by the mechanism in interpolated moves. If CX_ALL_SOLN (-1) is set as the parameter for soln, all solutions are searched and the closest one is set. If a number is entered that is 0 or above, the inverse kinematics solution should return a specific solution.

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>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_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>The mechanism is not opened for control.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>The specified node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The node is cut out and cannot be modified.</td>
</tr>
<tr>
<td>INVALID_NODE_ID</td>
<td>The node number does not match node ID.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The mechanism does not have the ROBOT attribute.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following example sets the default solution for the mechanism to be CX_ALL_SOLN. This forces the mechanism to use the closest solution.

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

void main(void)
{
    CxMechanism mech;
    CxNodeId   MyMech;
    long      soln_used;

    Server = CxOpenServer( "Testing", CX_SMEM,
                             CxGetNamedNodeId( Server, "MyMech", &MyMech );
    mech = CxOpenMechanism( Server, MyMech, CX_CONTROL );
```
CxSetDefaultSoln( mech, MyMech, CX_ALL_SOLN );
.
.
CxRobpacExit ();
}

SEE ALSO
CxGetDefaultSoln
**CxSetFclassNumber**

Sets a mechanism’s forward kinematics classification (fclass) number

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxSetFclassNumber(CxMechanism mech, CxNodeId mech_id, long class)
```

**ARGUMENTS**

- **mech**: The mechanism ID
- **mech_id**: The mechanism node ID
- **class**: The forward kinematics classification (fclass) number

**DESCRIPTION**

This function sets the forward kinematics classification (fclass) number for a given mechanism. The fclass number is used to call a user-supplied routine that determines dependent joint values and independent joint limits for mechanisms having more complex kinematics. The fclass number cannot be set to a value less than 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>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 given mechanism node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The node is cut out and cannot be modified.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the ROBOT attribute.</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>The class number entered is less than 0.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>The mechanism is not opened for control.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`CxGetFclassNumber`, `CxGetFwrdkinType`, `CxSetFwrdkinType`
CxSetFwdkinType
Sets a mechanism’s forward kinematics solution type

SYNOPSIS
#include <code/robpac.h>
long CxSetFwdkinType(CxMechanism mech, CxNodeId mech_id, long type)

ARGUMENTS
mech The mechanism ID
mech_id The mechanism node ID
type The mechanism forward kinematics type (CX_FUNCTION or CX_LIN_DEP)

DESCRIPTION
This function sets the forward kinematics type for a given mechanism. The allowable values are either
CX_INDEPEND (only independent joints present), CX_FUNCTION (some joints are functionally dependent
on others) or CX_LIN_DEP (some joints are linearly dependent on others).

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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The node is cut out and cannot be modified.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the Robot attribute.</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>The type entered is not CX_INDEPEND, CX_FUNCTION, or CX_LIN_DEP.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>The mechanism is not opened for control.</td>
</tr>
</tbody>
</table>

EXAMPLE
The following example gets the forward kinematics solution type used for the mechanism and prints it out. It
then sets the forward kinematics solution type to CX_FUNCTION, since the mechanism has a joint which is
functionally dependent on another joint of the mechanism.

#include <code/robpac.h>

void main(void)
{

CxServer    Server;
CxMechanism mech;
CxNodeId   MyMech;
long      soln_type;

Server = CxOpenServer( "Testing", CX_SMEM, 0 );
CxGetNamedNodeId( Server, "MyMech", &MyMech );
mech = CxOpenMechanism( Server, MyMech, CX_CONTROL );

CxGetFwrdkinType( mech, MyMech, &soln_type );
printf( " MyMech's forward kinematics solution type is: ");
if ( soln_type == CX_INDEPEND )
   printf( " independent\n " );
else if ( soln_type == CX_FUNCTION )
   printf( " functionally dependent\n " );
else if ( soln_type == CX_LIN_DEP )
   printf( " linearly dependent\n " );
else
   printf( " unknown\n " );

CxSetFwrdkinType ( mech, MyMech, CX_FUNCTION );
CxRobpacExit();

SEE ALSO
    CxGetFwrdkinType, CxGetFclassNumber, CxSetFclassNumber
**CxSetIclassNumber**

Sets a mechanism’s inverse kinematics classification (iclass) number

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxSetIclassNumber(CxMechanism mech, CxNodeId mech_id, long class)
```

**ARGUMENTS**

- `mech`: The mechanism ID
- `mech_id`: The mechanism node ID
- `class`: The inverse kinematics class (iclass) number

**DESCRIPTION**

This function sets the inverse kinematics classification (iclass) number for a given mechanism. The iclass number is used by the CIMServer to access a user-supplied custom inverse kinematics routine for a particular mechanism(s). The iclass number cannot be set to a value less than 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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The node is cut out and cannot be modified.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the ROBOT attribute.</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>The class number entered is less than 0.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>The mechanism is not opened for control.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

CxGetIclassNumber, CxGetInvkinType, CxSetInvkinType
**CxSetInvkinType**

Sets a mechanism’s inverse kinematics solution type

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxSetInvkinType(CxMechanism mech, CxNodeId mech_id, long type)
```

**ARGUMENTS**

- `mech`: The mechanism ID
- `mech_id`: The mechanism node ID
- `type`: The inverse kinematics solution type. See the list in the description below.

**DESCRIPTION**

This function sets the mechanism inverse kinematics type for a given mechanism. The following are the allowable types:

- `CX_AUTO_INVKIN`: CX_AUTO_DH procedures
- `CX_CUSTOM_INVKIN`: User-supplied function
- `CX_NUMERICAL_INVKIN`: Numerical inverse kinematics procedures
- `CX_NO_INVKIN`: No inverse kinematics procedures

**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><code>CX_MESSAGE_SEND_FAILED</code></td>
<td>An error in sending the message.</td>
</tr>
<tr>
<td><code>CX_MESSAGE_RECEIVE_FAILED</code></td>
<td>An error in receiving the message.</td>
</tr>
<tr>
<td><code>CX_NODE_NOT_FOUND</code></td>
<td>The given mechanism node does not exist.</td>
</tr>
<tr>
<td><code>CX_NODE_IS_CUT_OUT</code></td>
<td>The node is cut out and cannot be modified.</td>
</tr>
<tr>
<td><code>CX_ATTRIBUTE_NOT_FOUND</code></td>
<td>The given node does not have the ROBOT attribute.</td>
</tr>
<tr>
<td><code>CX_INVALID_ARGUMENT</code></td>
<td>The type entered is not <code>CX_AUTO_INVKIN</code>, <code>CX_CUSTOM_INVKIN</code>, <code>CX_NUMERICAL_INVKIN</code>, or <code>CX_NO_INVKIN</code>.</td>
</tr>
<tr>
<td><code>CX_INVALID_MECHANISM</code></td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td><code>CX_MECH_NOT_OPEN_FOR_CONTROL</code></td>
<td>The mechanism is not opened for control.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`CxGetInvkinType`, `CxGetIclassNumber`, `CxSetIclassNumber`
**CxSetJclassNumber**

Sets a mechanism’s joint rate classification (jclass) number

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxSetJclassNumber(CxMechanism mech, CxNodeId mech_id, long jclass)
```

**ARGUMENTS**

- **mech**  
The mechanism ID of mechanism open for control
- **mech_id**  
The mechanism node ID
- **jclass**  
The mechanism joint rate classification number

**DESCRIPTION**

This function sets the currently specified custom joint rate classification (jclass) number for a given mechanism. The jclass number is used by the CIMServer to call a user-supplied routine to calculate joint rates, given trajectory following of motion segments at specified speeds. The jclass number cannot be set to a value less than 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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The node is cut out and cannot be modified</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the ROBOT attribute.</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>The type entered is less than 0.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>The mechanism is not opened for control.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

CxGetJclassNumber
**CxSetJnt2actMap**

Sets the joint to actuator space mapping number

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxSetJnt2actMap (CxMechanism mech, long jnt2act_map)
```

**ARGUMENTS**

| mech         | The mechanism ID
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This is a CxMechanism ID returned from a call to CxOpenMechanism</td>
</tr>
</tbody>
</table>

| jnt2act_map  | The joint space to actuator space mapping number |

**DESCRIPTION**

This function allows the user to set a mapping number which is used to call a routine to convert between the joint space and the actuator space for the defined mechanism. The default value of `jnt2act_map` is 0, which assumes the joint value and the actuator value are the same for each joint.

**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>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_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

If a user wants to drive the puma253 junior robot, then they will make the following function call:

```c
CxSetJnt2actMap (mech, 2)
```

This will make the server use case 2 of user_motion.AJ and user_motion.JA files when converting actuator to joint or vice versa mapping respectively.

**Note:** In this example `q->dof` corresponds to values of the independent joints. `q->dof_act` corresponds to the mechanism’s actuator values.

Following is the C code for case 2 in user_motion.AJ; which describes the mapping from actuator space to joint space:

```c
case 2: /*** puma260, or puma253 (Junior) ***/
q->dof[0] = q->dof_act[0];
q->dof[1] = q->dof_act[1];
q->dof[2] = q->dof_act[2];
q->dof[3] = q->dof_act[3];
q->dof[5] = q->dof_act[5] + 0.176271*(q->dof[4]) + 0.031477*
```
Following is the code for case 2 in `user_motion.JA`. This describes the mapping from joint space to actuator space:

```c
(q->dof[3]);
break;

case 2: /*** puma260, or puma253 (Junior /***/
    for (i=0;i<4;i++)
    {
        q->dof_act[i] = q->dof[i];
        q->curspd_act[i] = q->curspd[i];
    }
    q->dof_act[5] = q->dof[5] - 0.176271* (q->dof[4]) - 0.031477*
                     q->dof[3];
                     - 0.031477* q->curspd[3];
                      - 0.031477 * q->curaccel[3];
break;

WARNINGS

If \texttt{jnt2act\_map} is set to a non-zero value, the user has to make sure that the appropriate code segments are added in files `user\_motion.JA` and `user\_motion.AJ` which are found in the `lib/cimetrix/custom` directory. See Chapter 5 of the \textit{Customizing CODE} manual for more information.
**CxSetMaxScrewSpeed**

Sets a mechanism’s maximum screw speed

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxSetMaxScrewSpeed(CxMechanism mech, CxNodeId mech_id, double speed)
```

**ARGUMENTS**

- **mech**  
The mechanism ID
- **mech_id**  
The mechanism node ID
- **speed**  
The maximum screw speed (deg/sec or rad/sec, depending on units. The default unit is degrees. See CxSetUnit in the Nodes, Frames and Attributes section of the CODE API Programmer’s Reference Manual- Volume 2 for more information).

**DESCRIPTION**

This function sets the maximum screw speed allowed for a given mechanism. The actual screw speed may be less than this value for all 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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The node is cut out and cannot be modified.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the ROBOT attribute.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>The mechanism is not opened for control.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following example gets the maximum screw speed and sets it to a new value as appropriate. It then continuously monitors its value during the motion. When moving from one point to another, two types of speeds are involved: a linear speed to reach from one point to another, and an angular speed to make the two frames align with each other (assuming that the tool motion type is CX_FULL_POSE). The angular speed is set by these 'screw' functions. During the motion, both the angular and the linear speeds are taken into account and whichever takes the maximum time governs the motion.

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

void main(void)
{
```
Server = CxOpenServer( "Testing", CX_SMEM, 0 );
CxGetNamedNodeId( Server, "MyMech", &MyMech );
CxGetNamedNodeId( Server, "tcf", &tcf );
CxGetNamedNodeId( Server, "t1", &t1 );

mech = CxOpenMechanism( Server, MyMech, CX_CONTROL );
CxSetBlendPolicy( mech, CX_MOVE_TO);
CxSetInterpType( mech, CX_LINEAR_INTERP );

CxGetMaxScrewSpeed( mech, MyMech, &max_speed );
/* Set motion parameters */
if (increase_speed_limit)
    CxSetMaxScrewSpeed( mech, MyMech, max_speed*1.1 );
else if (decrease_speed_limit)
    CxSetMaxScrewSpeed( mech, MyMech, max_speed*0.9 );
/* Move MyMech to accomplish a task */
CxMoveToNode( mech, t1, tcf );
CxMoveToNode( mech, t2, tcf );
CxMoveToNode( mech, t3, tcf );

while (1)
{
    CxGetActualScrewSpeed( mech, MyMech, &screw_speed );
    printf(" Current screw speed : %f \n", screw_speed );
    .
}

CxWaitForEndOfMotion( mech );
CxRobpacExit();

SEE ALSO
CxGetMaxScrewSpeed, CxGetScrewSpeed, CxSetScrewSpeed, CxGetActualScrewSpeed
**CxSetMaxToolSpeed**

Sets a mechanism’s maximum tool speed

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxSetMaxToolSpeed(CxMechanism mech, CxNodeId mech_id, double speed)
```

**ARGUMENTS**

- `mech`: The mechanism ID of mechanism open for control
- `mech_id`: The mechanism node ID
- `speed`: The maximum tool speed (in/sec, mm/sec, etc.)

**DESCRIPTION**

This function sets the maximum tool speed for a given mechanism. The actual tool speed may be less than this value in actual moves, due to joint rate limiting.

**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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The node is cut out and cannot be modified</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the ROBOT attribute.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>The mechanism is not opened for control.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following example gets the maximum tool speed and sets it to a new value as appropriate. It then continuously monitors its value during the motion. When moving from one point to another, two types of speeds are involved: a linear speed to reach from one point to another, and an angular speed to make the two frames align with each other (assuming that the tool motion type is CX_FULL_POSE). The linear speed is set by these 'tool' functions. During the motion, both the angular and the linear speeds are taken into account. Whichever takes the maximum time governs the motion.

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

void main(void)
{
```
CXServer    Server;
CXMechanism mech;
CxNodeId   MyMech, tcf, t1, t2, t3;
double    max_speed, tool_speed, screw_speed;
long      increase_speed_limit, decrease_speed_limit;

Server = CxOpenServer( "Testing", CX_SMEM, 0 );
CxGetNamedNodeId( Server, "MyMech", &MyMech );
CxGetNamedNodeId( Server, "tcf", &tcf );
CxGetNamedNodeId( Server, "t1", &t1 );
.
.
mech = CxOpenMechanism( Server, MyMech, CX_CONTROL );
CxSetBlendPolicy( mech, CX_MOVE_TO);
CxSetInterpType( mech, CX_LINEAR_INTERP );

CxGetMaxToolSpeed( mech, MyMech, &max_speed );
    /* Set motion parameters */
    if (increase_speed_limit)
        CxSetMaxToolSpeed( mech, MyMech, max_speed*1.1 )
    else if (decrease_speed_limit)
        CxSetMaxToolSpeed( mech, MyMech, max_speed*0.9 );

    /* Move MyMech to accomplish a task */
    CxMoveToNode( mech, t1, tcf );
    CxMoveToNode( mech, t2, tcf );
    CxMoveToNode( mech, t3, tcf );

    while (1)
    {
        CxGetActualToolSpeed( mech, MyMech, &tool_speed );
        printf(" Current tip speed : %f \n", tool_speed );
        .
    }

    CxWaitForEndOfMotion( mech );
    CxRobpacExit();
}

SEE ALSO

CxGetMaxToolSpeed,CxGetTipSpeed,CxSetTipSpeed,CxGetActualToolSpeed
**CxSetRclassNumber**

Sets a mechanism’s redundant classification (rclass) number

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxSetRclassNumber(CxMechanism mech, CxNodeId mech_id, long class)
```

**ARGUMENTS**

- **mech** The mechanism ID of mechanism open for control
- **mech_id** The mechanism node ID
- **class** The mechanism redundant classification (rclass) number

**DESCRIPTION**

This function sets the redundant classification (rclass) number for a given mechanism. The rclass number is used to call a user-supplied routine that supplies weighting factors to the joints in the CIMServer’s numerical inverse kinematics routine. The numerical inverse kinematics routine should only be used for simulation purposes. The rclass number cannot be set to a value less than 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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The node is cut out and cannot be modified</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the ROBOT attribute.</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>The class number entered is less than 0.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>The mechanism is not opened for control.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

CxGetRclassNumber
**CxSetScrewAccelMax**

Sets a mechanism’s maximum screw acceleration and deceleration

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxSetScrewAccelMax(CxMechanism mech, CxNodeId node, double accel_rise, double accel_fall)
```

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Arg</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mech</td>
<td>The mechanism ID or mechanism open for control</td>
</tr>
<tr>
<td>CxNodeId</td>
<td>The mechanism node ID</td>
</tr>
<tr>
<td>accel_rise</td>
<td>The maximum screw acceleration</td>
</tr>
<tr>
<td>accel_fall</td>
<td>The maximum screw deceleration (deg/sec or rad/sec, depending on units. The default unit is degrees. See CxSetUnit in the Nodes, Frames and Attributes section of the CODE API Programmer’s Reference Manual- Volume 2 for more information.</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

The function allows the user to set the maximum screw (rotational) acceleration and deceleration values for Cartesian motion where the primary motion is rotation about a space axis.

**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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The node is cut out and cannot be modified</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the ROBOT attribute.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>The mechanism is not opened for control.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following example gets the maximum trap acceleration and deceleration and modifies them as appropriate. When moving from one point to another, two types of velocities are involved: a linear velocity required to reach from one point to another, and an angular velocity to make the two frames align with each other (assuming that the tool motion type is CX_FULL_POSE). When the acceleration type equals CX_CONST_RAMP_ACCEL, the angular acceleration and deceleration is set by these ‘screw’ functions.

```c
#include <code/robpac.h>
```
void main(void)
{
    CxServer   Server;
    CxMechanism mech;
    CxNodeId   MyMech, tcf, t1, t2, t3;
    double     rise_time, fall_time;
    double     accel, decel;
    long       increase_accel_limit, decrease_accel_limit;

    Server = CxOpenServer( "Testing", CX_SMEM, 0 );
    CxGetNamedNodeId( Server, "MyMech", &MyMech );
    CxGetNamedNodeId( Server, "tcf", &tcf );
    CxGetNamedNodeId( Server, "t1", &t1 );

    mech = CxOpenMechanism( Server, MyMech, CX_CONTROL );
    CxSetBlendPolicy( mech, CX_MOVE_TO);
    CxSetInterpType( mech, CX_LINEAR_INTERP );
    CxSetAccelType( mech, CX_CONST_RAMP_ACCEL );

    CxGetScrewAccelMax( mech, MyMech, &accel, &decel );
    /* Set motion parameters */
    if (increase_accel_limit)
        CxSetScrewAccelMax( mech, MyMech, accel*1.1, decel*1.1 );
    else if (decrease_accel_limit)
        CxSetScrewAccelMax( mech, MyMech, accel*0.9, decel*0.9 );

    /* Move MyMech to accomplish a task */
    CxMoveToNode( mech, t1, tcf );
    CxMoveToNode( mech, t2, tcf );
    CxMoveToNode( mech, t3, tcf );

    CxWaitForEndOfMotion( mech );
    CxRobpacExit();
}

WARNINGS

These values will limit the Cartesian rise and fall rotational accelerations specified when using other accel API functions, if these values exceed the maximums. Since allowable maximums are mechanism- and design-dependent, the user must be extremely cautious when entering these values if they will be used to actually control a mechanism. Excessive accelerations may cause the mechanism’s inertial loads to damage the drive transmissions and other mechanical components on the mechanism.

SEE ALSO

CxGetScrewAccelMax, CxGetAccelType, CxSetAccelType
**CxSetTclassNumber**

Sets a mechanism’s trajectory classification (tclass) number

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxSetTclassNumber(CxMechanism mech, CxNodeId mech_id, long class)
```

**ARGUMENTS**

- **mech**
  The mechanism ID of mechanism open for control
- **mech_id**
  The mechanism ID
- **class**
  The trajectory classification (tclass) number

**DESCRIPTION**

This function sets a mechanism’s trajectory classification (tclass) number, which is used to select a trajectory generation scheme. The default setting for the class number is 0, which selects the software trajectory generation scheme. The user can supply a customized trajectory generation scheme by modifying the file `%ROBOT%\lib\cimetrix\custom\user_motion.FG`; however, 0-999 are reserved for trajectory generators developed by Cimetrix. The user can use any number that is greater than 999 for a user-supplied trajectory generator.

**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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The node is cut out and cannot be modified</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the ROBOT attribute.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>The mechanism is not opened for control.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

CxSetTclassNumber
**CxSetTifFrame**

Sets the terminal interface frame

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxSetTifFrame(CxMechanism mech, CxNodeId mech_id, CxNodeId frame)
```

**ARGUMENTS**

- `mech` The mechanism ID of mechanism open for control
- `mech_id` The mechanism node ID
- `frame` Mechanism’s terminal interface frame node ID

**DESCRIPTION**

This function sets the terminal interface frame for the given mechanism. Rather than using the last joint frame in the inverse kinematics calculations, manufacturers sometimes decides to use an interface frame which corresponds to the mounting interface for tools on mechanisms.

**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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The node is cut out and cannot be modified.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the ROBOT attribute.</td>
</tr>
<tr>
<td>CX_TIF_NOT_FOUND</td>
<td>The tif node does not exist.</td>
</tr>
<tr>
<td>CX_TIF_NOT_A_CHILD_OF_LAST_JNT</td>
<td>The TIF node does not exist or TIF node is not a child of the last joint frame.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_ROBOT_HAS_NO_JOINTS</td>
<td>The mechanism does not have joints.</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>The mechanism is not opened for control.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following example sets the TIF to be used by the mechanism for a motion.

```c
#include <code/robpac.h>
```
void main(void)
{
    CxServer    Server;
    CxMechanism mech;
    CxNodeId   MyMech, tcf;

    Server = CxOpenServer( "Testing", CX_SMEM, 0 );
    CxGetNamedNodeId( Server, "MyMech", &MyMech );
    CxGetNamedNodeId( Server, "tcf", &tcf );
    mech = CxOpenMechanism( Server, MyMech, CX_CONTROL );

    CxSetTifFrame( mech, MyMech, tcf );

    CxRobpacExit ();
}

SEE ALSO
    CxGetTifFrame
CxSetTrapAccelMax

Sets the max rise and fall accels for Cartesian path following

SYNOPSIS

```c
#include <code/robpac.h>
long CxSetTrapAccelMax(CxMechanism mech, CxNodeId mech_id, double accel_rise, double accel_fall)
```

ARGUMENTS

- `mech` The mechanism ID of mechanism open for control
- `mech_id` The mechanism node ID
- `accel_rise` The maximum positive acceleration value in user units (mm/s^2, in/s^2, etc.)
- `accel_fall` The maximum positive deceleration value in user units (mm/s^2, in/s^2, etc.)

DESCRIPTION

The function allows the user to set the maximum allowable rise and fall accelerations for trajectory following in Cartesian space. These should be positive values in the current user 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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The node is cut out and cannot be modified.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the ROBOT attribute.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>The mechanism is not opened for control.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following example gets the maximum trap acceleration and deceleration and modifies them as appropriate. When moving from one point to another, two types of velocities are involved: a linear velocity required to reach from one point to another, and an angular velocity to make the two frames align with each other (assuming that the tool motion type is CX_FULL_POSE). When the acceleration type equals CX_CONST_RAMP_ACCEL, the linear acceleration and deceleration is set by these 'trap' functions.

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

void main(void)
```
{  
  CxServer    Server;
  CxMechanism mech;
  CxNodeId   MyMech, tcf, t1, t2, t3;
  double    accel, decel;
  long       increase_accel_limit, decrease_accel_limit;

  Server = CxOpenServer( "Testing", CX_SMEM, 0 );
  CxGetNamedNodeId( Server, "MyMech", &MyMech );
  CxGetNamedNodeId( Server, "tcf", &tcf );
  CxGetNamedNodeId( Server, "t1", &t1 );

  mech = CxOpenMechanism( Server, MyMech, CX_CONTROL );
  CxSetBlendPolicy( mech, CX_MOVE_TO);
  CxSetInterpType( mech, CX_LINEAR_INTERP );
  CxSetAccelType( mech, CX_CONST_RAMP_ACCEL );

  CxGetTrapAccelMax( mech, MyMech, &accel, &decel );
  /* Set motion parameters */
  if (increase_accel_limit)
      CxSetTrapAccelMax( mech, MyMech, accel*1.1, decel*1.1);
  else if (decrease_accel_limit)
      CxSetTrapAccelMax( mech, MyMech, accel*0.9, decel*0.9 );

  /* Move MyMech to accomplish a task */
  CxMoveToNode( mech, t1, tcf );
  CxMoveToNode( mech, t2, tcf );
  CxMoveToNode( mech, t3, tcf );

  CxWaitForEndOfMotion( mech );
  CxRobpacExit();
}

WARNINGS
Since allowable maximums are mechanism- and design-dependent, the user must be extremely cautious when entering these values if they will be used to actually control a mechanism. Excessive accelerations may cause the mechanism's inertial loads to damage the drive transmissions and other mechanical components on the mechanism.

SEE ALSO
CxGetTrapAccelMax, CxSetTrapAccelRamps, CxGetTrapAccelRamps,
CxGetAccelType, CxSetAccelType
Axis Parameters
**CxGetJntAccel**

Gets current joint acceleration and deceleration settings

**SYNOPSIS**

```
#include <code/robpac.h>
long CxGetJntAccel(CxMechanism mech, long jnt_number, double *accel_rise, double *accel_fall)
```

**ARGUMENTS**

- **mech**
  - The mechanism ID
- **jnt_number**
  - The independent joint number (count from 0)
- **accel_rise**
  - Positive joint acceleration (if it is a revolute joint, the unit is in deg/sec^2 or rad/sec^2, depending on unit. The default unit is degrees. See CxSetUnit in the Nodes, Frames and Attributes section of the CODE API Programmer's Reference Manual - Volume 2 for more information).
- **accel_fall**
  - Positive joint deceleration (if it is a revolute joint, the unit is set in deg/sec^2 or rad/sec^2, depending on unit. The default unit is degrees. See CxSetUnit in the Nodes, Frames and Attributes section of the CODE API Programmer's Reference Manual - Volume 2 for more information).

**DESCRIPTION**

This function allows the user to get the current joint acceleration and deceleration values.

**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>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_INVALID_JOINT_NUMBER</td>
<td>The value entered is not within independent joint range of robot.</td>
</tr>
<tr>
<td>CX_JOINT_NOT_FOUND</td>
<td>The specified joint does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The given node is cut out.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following example performs the following tasks for each joint of a mechanism. First, it gets the current maximum acceleration and deceleration settings and the actual acceleration and deceleration values used for the joint movement. Next, it checks a software flag to see if the mechanism is set to run slowly or not. Finally, it adjusts the acceleration and deceleration settings based on the flag and the acceleration limits.

```
#include <code/robpac.h>
#define TOT_JOINTS 6

void main (void)
{
    CxServer Server;
```
CxMechanism mech;
CxNodeId robot;
long jnt_nr;
double cur_accel, cur_decel;
double max_accel, max_decel;
long run_slow;

Server = CxOpenServer( "Testing", CX_SMEM, 0 );
CxGetNamedNodeId( Server, "robot", &robot );
mech = CxOpenMechanism( Server, robot, CX_CONTROL );

for (jnt_nr=0; jnt_nr<TOT_JOINTS; jnt_nr++) {
    /* Get current maximum and running accel and decel settings */
    CxGetJntAccel_Max( mech, jnt_nr, &max_accel, &max_decel );
    CxGetJntAccel( mech, jnt_nr, &cur_accel, &cur_decel );

    /* Set robot acceleration and deceleration values depending on run mode */
    if ( run_slow )
        if ((cur_accel>max_accel*0.1) || (cur_decel>max_decel*0.1))
            CxSetJntAccel( mech, jnt_nr, max_accel*0.1,
                           max_decel*0.1 );
    else
        if ((cur_accel>max_accel*0.3) || (cur_decel>max_decel*0.3))
            CxSetJntAccel( mech, jnt_nr, max_accel*0.3, max_decel*0.3);
}

CxRobpacExit();

WARNINGS
Since allowable maximums are robot- and design-dependent, the user must be extremely cautious when using these values to actually control a robot. Excessive accelerations may cause the robot’s inertial loads to damage the drive transmissions and other mechanical components on the robot.

SEE ALSO
CxGetTrapAccelTimes, CxGetJntAccelMax, CxSetJntAccel, CxGetAccelType, CxSetAccelType
CxGetJntAccelMax

Gets the maximum joint acceleration and deceleration values

SYNOPSIS

#include <code/robpac.h>
long CxGetJntAccel_max(CxMechanism mech, long jnt_number, double *accel_rise_max, double *accel_fall_max)

ARGUMENTS

mech                The mechanism ID
jnt_number          The independent joint number (count from 0)
accel_rise_max      Positive max joint acceleration (if it is a revolute joint, the unit is deg/sec^2 or rad/sec^2, depending on unit. The default unit is degrees. See CxSetUnit in the Nodes, Frames and Attributes section of the CODE API Programmer's Reference Manual- Volume 2 for more details).
accel_fall_max      Positive max joint deceleration (if it is a revolute joint, the unit is deg/sec^2 or rad/sec^2, depending on unit. The default is degrees. See CxSetUnit in the Nodes, Frames and Attributes section of the CODE API Programmer's Reference Manual- Volume 2 for more information).

DESCRIPTION

This function allows the user to get current maximum joint acceleration and deceleration values.

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>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_INVALID_JOINT_NUMBER</td>
<td>The value entered is not within independent joint range of robot.</td>
</tr>
<tr>
<td>CX_JOINT_NOT_FOUND</td>
<td>The specified joint does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The given node is cut out.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following example performs the following tasks for each joint of a mechanism. First, it gets the current maximum acceleration and deceleration settings and the actual acceleration and deceleration values used for the joint movement. Next, it checks a software flag to see if the mechanism is set to run slowly or not. Finally, it adjusts the acceleration and deceleration settings based on the flag and the acceleration limits.

#include <code/robpac.h>
#define TOT_JOINTS 6
void main (void) {
    CxServer    Server;
    CxMechanism mech;
    CxNodeId   robot;
    long       jnt_nr;
    double     cur_accel, cur_decel;
    double     max_accel, max_decel;
    long       run_slow;

    Server = CxOpenServer( "Testing", CX_SMEM, 0 );
    CxGetNamedNodeId( Server, "robot", &robot );
    mech = CxOpenMechanism( Server, robot, CX_CONTROL );

    for  (jnt_nr=0; jnt_nr<TOT_JOINTS; jnt_nr++) {
        /* Get current maximum and running accel and decel settings */
        CxGetJntAccelMax( mech, jnt_nr, &max_accel, &max_decel );
        CxGetJntAccel( mech, jnt_nr, &cur_accel, &cur_decel );

        /* Set robot acceleration and deceleration values depending on run mode */
        if ( run_slow ) {
            if ( (cur_accel>max_accel*0.1) || (cur_decel>max_decel*0.1) )
                CxSetJntAccel( mech, jnt_nr, max_accel*0.1, max_decel*0.1 );
        }
        else {
            if ((cur_accel>max_accel*0.3) || (cur_decel>max_decel*0.3))
                CxSetJntAccel( mech, jnt_nr, max_accel*0.3, max_decel*0.3 );
        }
    }

    CxRobpacExit();
}

WARNINGS
Since allowable maximums are robot- and design-dependent, the user must be extremely cautious when using these values to actually control a robot. Excessive accelerations may cause the robot’s inertial loads to damage the drive transmissions and other mechanical components on the robot.

HARDWARE AND SYSTEM DEPENDENCIES
The numbers are the effective acceleration rates allowed, considering the joint transmission mechanisms and the effective inertia of all links which the motor must accelerate.

SEE ALSO
CxGetTrapAccelTimes, CxSetJntAccelMax, CxGetJntAccel, CxSetJntAccel
CxGetJntDepConst

Gets the joint dependency parameter for a linearly dependent joint

SYNOPSIS

#include <code/robpac.h>
long CxGetJntDepConst(CxMechanism mech, long dep_jnt_number, long *indep_jnt_number, double *dconst)

ARGUMENTS

mech
The mechanism ID
dep_jnt_number
The linearly dependent joint number
indep_jnt_number
The joint number of the independent joint upon which the dependent joint depends
dconst
The linear dependency constant

DESCRIPTION

This function is used to get the independent joint number and linear dependency constant for the specified dep_jnt_number. The linear dependency constant relates the dependent joint to the independent joint as follows: 
Value of dependent joint = linear dependency constant × value of independent joint.

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>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_JOINT_NOT_FOUND</td>
<td>The specified joint does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_JOINT_NUMBER</td>
<td>The Joint number entered is not valid.</td>
</tr>
<tr>
<td>CX_JOINT_NOT_LINEAR_DEP</td>
<td>Joint is not of type CX_LIN_DEP.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The given node is cut out.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following example first opens a mechanism, then performs the following task for each joint within that mechanism. First, it gets the joint dependency type. If the current joint is of type CX_LIN_DEP, the code gets the dependency constant and the node number on which the current node depends. Next, it prints out this information. Finally, depending on the selected gear ratio, it resets the dependency ratio to either 0.5 or 3.0.

```c
#include <stdio.h>
#include <code/robpac.h>
#define  TOT_JOINTS  6
```
void main (void) {
    CxServer    Server;
    CxMechanism mech;
    CxNodeId    robot;
    long        dependency_type;
    double      depend_const;
    long        jnt_nr;
    long        lower_gear_ratio;
    long        indep_jnt_nr;
    
    Server = CxOpenServer( "Testing", CX_SMEM 0 );
    CxGetNamedNodeId( Server, "robot", &robot );
    
    mech = CxOpenMechanism( Server, robot, CX_CONTROL );
    for (jnt_nr=0; jnt_nr<TOT_JOINTS; jnt_nr++) {
        CxGetJntDepType( mech, jnt_nr, &dependency_type );
        if ( dependency_type == CX_LIN_DEP ) {
            CxGetJntDepConst( mech, jnt_nr, &indep_jnt_nr, &depend_const );
            printf(" Joint #%ld is Linearly dependent on Joint #%ld\n",
                    jnt_nr, indep_jnt_nr);
            printf(" Dependency factor is %lf \n", depend_const);
            if (lower_gear_ratio)
                CxSetJntDepConst( mech, jnt_nr, indep_jnt_nr, 0.5 );
            else
                CxSetJntDepConst( mech, jnt_nr, indep_jnt_nr, 3.0 );
        }
    }
    
    CxRobpacExit();
}

SEE ALSO
    CxSetJntDepConst, CxSetJntDepType
CxGetJntDepType

Gets the joint dependency type

SYNOPSIS

#include <code/robpac.h>
long CxGetJntDepType(CxMechanism mech, long jnt_number, long *type)

ARGUMENTS

mech The mechanism ID
jnt_number The Joint number (count from 0)
type The joint's dependency type

DESCRIPTION

This function is used to get the dependency type for the given joint. The value returned for type will be one of the following:

• CX_INDEPEND
• CX_LIN_DEP
• CX_FUNCTION

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>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_JOINT_NOT_FOUND</td>
<td>The specified joint does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_JOINT_NUMBER</td>
<td>The Joint number entered is not valid.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The given node is cut out.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following example opens a mechanism, then performs the following task for each joint within that mechanism. First, it gets the joint dependency type. If the current joint is of type CX_LIN_DEP, the code gets the dependency constant and the node number on which the current node depends. Next, it prints out this information. Finally, depending on the selected gear ratio, it resets the dependency ratio to either 0.5 or 3.0.

#include <stdio.h>
#include <code/robpac.h>

#define TOT_JOINTS 6

void main (void)
{ 
    CxServer    Server;
    CxMechanism mech;
    CxNodeId   robot;
    long      dependency_type;
    double    depend_const;
    long      jnt_nr;
    long      lower_gear_ratio;
    long      indep_jnt_nr;

    Server = CxOpenServer( "Testing", CX_SMEM, 0 );
    CxGetNamedNodeId( Server, "robot", &robot );
.
.
    mech = CxOpenMechanism( Server, robot, CX_CONTROL );
    for  (jnt_nr=0; jnt_nr<TOT_JOINTS; jnt_nr++) {
        CxGetJntDepType( mech, jnt_nr, &dependency_type );
        if ( dependency_type == CX_LIN_DEP )
            {
            CxGetJntDepConst( mech, jnt_nr, &indep_jnt_nr, &depend_const );
            printf(" Joint #%ld is Linearly dependent on Joint #%ld\n", jnt_nr, indep_jnt_nr );
            printf(" Dependency factor is %lf \n", depend_const );
            if (lower_gear_ratio)
                CxSetJntDepConst( mech, jnt_nr, indep_jnt_nr, 0.5 );
            else
                CxSetJntDepConst( mech, jnt_nr, indep_jnt_nr, 3.0 );
            }
    }
.
.
    CxRobpacExit();
}

SEE ALSO
CxSetJntDepType, CxGetJntDepConst, CxSetJntDepConst
**CxGetJntInfinite**

Determines if a rotational joint has infinite joint limits

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxGetJntInfinite(CxMechanism mech, long jnt_number, long *flag)
```

**ARGUMENTS**

- `mech` The mechanism ID
- `jnt_number` The independent rotational joint number (count from 0)
- `flag` Flag set to either `CX_TRUE` (infinite joint) or `CX_FALSE` (finite limit) for joint having finite limits for infinite joint

**DESCRIPTION**

This function determines whether the joint limits of a rotational joint are unbounded by returning an argument value of `CX_TRUE`, or bounded by returning an argument value of `CX_FALSE`. A translational joint may not be infinite.

**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>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_JOINT_NOT_FOUND</td>
<td>The specified joint does not exist.</td>
</tr>
<tr>
<td>CX_WRONG_TYPE_JOINT</td>
<td>The specified joint is not independent or is not a rotational joint.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The given node is cut out.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following example checks every joint in a mechanism to see if it is translational and if it has infinite limits. If so, it prints out an error message and exits the program.

```c
#include <stdio.h>
#include <code/robpac.h>
#define   TOT_JOINTS  6

void main (void)
{
   CxServer    Server;
   CxMechanism mech;
   // Code follows here...
}
```
CxNodeId robot, jnt6;
long jnt_nr, jnt_type, is_infinite_limit;

Server = CxOpenServer( "Testing", CX_SMEM, 0 );
CxGetNamedNodeId( Server, "robot", &robot );

mech = CxOpenMechanism( Server, robot, CX_CONTROL );

for (jnt_nr=0; jnt_nr<TOT_JOINTS; jnt_nr++)
{
    CxGetJntType( mech, jnt_nr, &jnt_type );
    CxGetJntInfinite( mech, jnt_nr, &is_infinite_limit );
    if ( (jnt_type == CX_TRANS) && (is_infinite_limit) )
    {
        printf(" Aborting! Translational joints cannot have
 infinite limit. This should never happen!\n");
        CxRobpacExit ();
    }
}

SEE ALSO
   CxMoveSingleAxis,CxMoveAllAxes
**CxGetJntNode**

Gets the joint node ID using the joint number

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxGetJntNode(CxMechanism mech, long jnt_number, CxNodeId *node)
```

**ARGUMENTS**

- **mech**
  - The mechanism ID
- **jnt_number**
  - The Joint number (count from 0)
- **node**
  - The joint node ID

**DESCRIPTION**

This function is used to get the node ID for the given joint number.

**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>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_INVALID_JOINT_NUMBER</td>
<td>The given joint number is not valid.</td>
</tr>
<tr>
<td>CX_JOINT_NOT_FOUND</td>
<td>The specified joint does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The given node is cut out.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following segment of code gets the joint node information for joint 1:

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

CxServer Server;
CxMechanism mech;
.
.
void main(void)
{
    CxNodeId jnt1;
    node_info jnt_info;
    
    CxGetJntNode( mech, 1, &jnt1);
    CxGetNodeInfo( Server, jnt1, &jnt1_info);
```
SEE ALSO

CxGetJntNumber, CxGetNodeName, CxGetJointNumber
**CxGetJntNumber**

Gets the joint number given the joint node ID

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxGetJntNumber(CxMechanism mech, CxNodeId node, long *jnt_number)
```

**ARGUMENTS**

- `mech`: The mechanism ID
- `node`: The joint node ID
- `jnt_number`: The joint number (count from 0)

**DESCRIPTION**

This function is used to get the CIMServer-assigned number for the given joint 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 Codes</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 given node does not exist.</td>
</tr>
<tr>
<td>CX_JOINT_NOT_FOUND</td>
<td>The given node does not have the JOINT attribute.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</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_IS_CUT_OUT</td>
<td>The given node is cut out.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>Node does not have joint attribute.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following segment of code gets the joint number for the node `left_finger` and moves the joint 10 degrees away from the home position:

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

do
void main(void)
{
    CxServer Server;
    CxMechanism mech;
    CxNodeId Mule, left_finger;
    long j_left_finger;

    Server = CxOpenServer ("TestServer", CX_SMEM, 0);
```
CxGetNamedNodeId (Server, "left_finger", &left_finger);
CxGetNamedNodeId (Server, "Mule", &Mule);
.
.
mech = CxOpenMechanism (Server, Mule, CX_CONTROL);
.
.
CxSetInterpType ( mech, CX_JOINT_INTERP);
CxGetJntNumber( mech, left_finger, &j_left_finger);

/* move left_finger 10 degrees */
CxMoveSingleAxis( mech, j_left_finger, 10.)
.
.
}

SEE ALSO
CxGetJntNode
**CxGetJntType**

Gets joint type (rotational or translational)

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxGetJntType(CxMechanism mech, long jnt_number, long *type)
```

**ARGUMENTS**

- `mech`: The mechanism ID
- `jnt_number`: The Joint number (starts from 0)
- `type`: The joint type (CX_ROTATE or CX_TRANS)

**DESCRIPTION**

This function is used to get the joint type (rotational or translational) for the given joint.

**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>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_NO_ROB_FOR_THE_JOINT</td>
<td>An invalid mechanism ID, or the mechanism does not have a robot assigned to it.</td>
</tr>
<tr>
<td>CX_JOINT_NOT_FOUND</td>
<td>The specified joint does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_JOINT_NUMBER</td>
<td>The Joint number entered is not valid.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The given node is cut out.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following example checks every joint in a mechanism to see if it is translational and if it has infinite joint limits. If so, it prints out an error message and exits the program.

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

#define TOT_JOINTS 6

void main (void)
{
    CxServer    Server;
    CxMechanism mech;
    CxNodeId   robot, jnt6;
    long      jnt_nr, jnt_type, is_infinite_limit;
```
Server = CxOpenServer( "Testing", CX_SMEM, 0 );
CxGetNamedNodeId( Server, "robot", &robot );
mech = CxOpenMechanism( Server, robot, CX_CONTROL );
.
.
for (jnt_nr=0; jnt_nr<TOT_JOINTS; jnt_nr++)
{
    CxGetJntType( mech, jnt_nr, &jnt_type );
    CxGetJntInfinite( mech, jnt_nr, &is_infinite_limit );
    if ( (jnt_type == CX_TRANS) && (is_infinite_limit) )
    {
        printf(" Aborting! Translational joints cannot have
                infinite limit. This should never happen!\n\n");
        doCleanup();
        CxRobpacExit();
    }
}
.
.
SEE ALSO

CxSetJntType
**CxGetMaxDofValue**

Gets maximum joint value (upper limit)

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxGetMaxDofValue(CxMechanism mech, long jnt_number, double *dof)
```

**ARGUMENTS**

- `mech` The mechanism ID
- `jnt_number` The Joint number (starts from 0)
- `dof` Used for upper limit of a joint value. For revolute joints, the units can be either degrees or radians. The default unit is degrees (see CxSetUnit in the Nodes, Frames and Attributes section of the CODE API Programmers Reference Manual- Volume 2 for details). For translational joints, this value is expressed in user units.

**DESCRIPTION**

This function is used to get the upper joint limit.

**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>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_JOINT_NOT_FOUND</td>
<td>The specified joint does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_JOINT_NUMBER</td>
<td>The Joint number entered is not valid.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The given node is cut out.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following example prints out the minimum and maximum values for each joint of a mechanism.

```c
#include <stdio.h>
#include <code/robpac.h>
#define TOT_JOINTS  6

void main (void) {
    CxServer    Server;
    CxMechanism mech;
    CxNodeId    robot;
    long        jnt_nr;
    double      min_value, max_value;
```
Server = CxOpenServer( "Testing", CX_SMEM, 0 );
CxGetNamedNodeId( Server, "robot", &robot );
mech = CxOpenMechanism( Server, robot, CX_CONTROL );

for (jnt_nr=0; jnt_nr<TOT_JOINTS; jnt_nr++)
{
    CxGetMinDofValue( mech, jnt_nr, &min_value );
    CxGetMaxDofValue( mech, jnt_nr, &max_value );
    printf(" Joint # : %ld, Min joint value : %lf, Max joint value :
            %lf \n",
            jnt_nr, min_value, max_value );
}

CxRobpacExit();

SEE ALSO
    CxSetMaxDofValue, CxGetMinDofValue, CxSetJntInfinite
**CxGetMaxJntSpeed**

*Gets the maximum joint speed*

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxGetMaxJntSpeed(CxMechanism mech, long jnt_number, double *speed)
```

**ARGUMENTS**

- `mech`: The mechanism ID
- `jnt_number`: The independent joint number (count from 0)
- `speed`: The maximum joint speed. For revolute joints, the units are in deg/sec or rad/sec, with the default being deg/sec (see `CxSetUnit` in the *Nodes, Frames and Attributes* section of the CODE API Programmers Reference Manual- Volume 2 for details). For translational joints, the speed is in units/sec.

**DESCRIPTION**

This function is used to get the maximum joint speed.

**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>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_JOINT_NOT_FOUND</td>
<td>The specified joint does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_JOINT_NUMBER</td>
<td>The Joint number entered is not valid.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The given node is cut out.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following example gets the maximum value set for each joint. Then, depending on the running mode, it increases or decreases the joint speed limit.

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

#define  TOT_JOINTS  6

void main (void)
{
    CxServer    Server;
    CxMechanism mech;
    CxNodeId   robot;
```
long    jnt_nr;
double    max_speed;
long    run_slower, run_faster;

Server = CxOpenServer( "Testing", CX_SMEM, 0 );
CxGetNamedNodeId( Server, "robot", &robot );
mech = CxOpenMechanism( Server, robot, CX_CONTROL );

for  (jnt_nr=0; jnt_nr<TOT_JOINTS; jnt_nr++) {
    CxGetMaxJntSpeed( mech, jnt_nr, &max_speed);
    if ( run_slower )
        CxSetMaxJntSpeed( mech, jnt_nr, max_speed*0.9 );
    else  if (run_faster )
        CxSetMaxJntSpeed( mech, jnt_nr, max_speed*1.1 );
}

CxRobpacExit();

SEE ALSO

CxSetMaxJntSpeed, CxGetJointSpeed, CxSetJointSpeed
## CxGetMinDofValue

Gets the minimum joint value (lower limit)

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxGetMinDofValue(CxMechanism mech, long jnt_number, double *dof)
```

**ARGUMENTS**

- `mech` The mechanism ID
- `jnt_number` The joint number (count from 0)
- `dof` The lower limit of a joint value. For revolute joints, the units can be either degrees or radians. The default unit is degrees (see `CxSetUnit` in the Nodes, Frames and Attributes section of the CODE API Programmer's Reference Manual- Volume 2 for more details). For translational joints, the value is expressed in user units.

**DESCRIPTION**

This function is used to get the lower joint limit.

**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>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_JOINT_NOT_FOUND</td>
<td>The specified joint does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_JOINT_NUMBER</td>
<td>The Joint number entered is not valid.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The given node is cut out.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following example prints out the minimum and maximum values for each joint of a mechanism.

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

#define TOT_JOINTS 6

void main (void)
{
    CxServer    Server;
    CxMechanism mech;
    CxNodeId    robot;
    long        jnt_nr;
    double      min_value, max_value;
```
Server = CxOpenServer( "Testing", CX_SMEM, 0 );
CxGetNamedNodeId( Server, "robot", &robot );
mech = CxOpenMechanism( Server, robot, CX_CONTROL );

for (jnt_nr=0; jnt_nr<TOT_JOINTS; jnt_nr++)
{
    CxGetMinDofValue( mech, jnt_nr, &min_value );
    CxGetMaxDofValue( mech, jnt_nr, &max_value );
    printf(" Joint # : %ld Min joint value : %lf Max joint value : %lf \n",
           jnt_nr, min_value, max_value );
}

CxRobpacExit();

SEE ALSO
    CxSetMinDofValue, CxGetMaxDofValue, CxSetJntInfinite
**CxSetJntAccel**

Sets the desired joint acceleration and deceleration values

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxSetJntAccel(CxMechanism mech, long jnt_number, double accel_rise, double accel_fall)
```

**ARGUMENTS**

- `mech` The mechanism ID of mechanism open for control
- `jnt_number` The independent joint number (count from 0)
- `accel_rise` Positive joint acceleration (if it is a revolute joint, the unit is in deg/sec\(^2\) or rad/sec\(^2\), depending on unit. The default unit is degrees. See `CxSetUnit` in the Nodes, Frames and Attributes section of the CODE API Programmer’s Reference Manual—Volume 2 for more details).
- `accel_fall` Positive joint deceleration (if it is a revolute joint, the unit is in deg/sec\(^2\) or rad/sec\(^2\), depending on unit. The default unit is degrees. See `CxSetUnit` in the Nodes, Frames and Attributes section of the CODE API Programmer’s Reference Manual—Volume 2 for more details).

**DESCRIPTION**

This function allows the user to set desired joint acceleration and deceleration values. If the CIMServer trajectory algorithms are used, and the desired value exceeds the maximum allowed for the robot, the value will internally be set to the currently specified maximum.

**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>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_JOINT_NOT_FOUND</td>
<td>The specified joint does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_JOINT_NUMBER</td>
<td>The Joint number entered is not valid.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>The mechanism is not opened for control.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The given node is cut out.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following example performs the following tasks for each joint of a mechanism. First, it gets the current maximum acceleration and deceleration settings and the actual acceleration and deceleration values used for the joint movement. Next, it checks a software flag to see if the mechanism is set to run slowly or not. Finally, it adjusts the acceleration and deceleration settings based on the flag and the acceleration limits.
#include <code/robpac.h>
#define  TOT_JOINTS  6

void main (void) {
    CxServer    Server;
    CxMechanism mech;
    CxNodeId   robot;
    long      jnt_nr;
    double    cur_accel, cur_decel;
    double    max_accel, max_decel;
    long      run_slow;

    Server = CxOpenServer( "Testing", CX_SMEM, 0 );
    CxGetNamedNodeId( Server, "robot", &robot );
    mech = CxOpenMechanism( Server, robot, CX_CONTROL );

    for  (jnt_nr=0; jnt_nr<TOT_JOINTS; jnt_nr++) {
        /* Get current maximum and running accel and decel settings */
        CxGetJntAccelMax( mech, jnt_nr, &max_accel, &max_decel );
        CxGetJntAccel( mech, jnt_nr, &cur_accel, &cur_decel );

        /* Set robot acceleration and deceleration values depending on 
         * run mode */
        if ( (cur_accel>max_accel*0.1) || (cur_decel>max_decel*0.1) )
            CxSetJntAccel( mech, jnt_nr, max_accel*0.1, max_decel*0.1 );
        else
            if ( (cur_accel>max_accel*0.3) || (cur_decel>max_decel*0.3) )
                CxSetJntAccel( mech, jnt_nr, max_accel*0.3, max_decel*0.3 );

    }

    CxRobpacExit();
}

WARNINGS
Since the allowable maximums are robot- and design-dependent, the user must be extremely cautious when using these values to actually control a robot. Excessive accelerations may cause the robot's inertial loads to damage the drive transmissions and other mechanical components on the robot.

HARDWARE AND SYSTEM DEPENDENCIES
The numbers entered are the effective acceleration rate allowed, considering the joint transmission mechanisms and the effective inertia of all links which the motor must accelerate.

SEE ALSO
CxGetTrapAccelTimes, CxGetJntAccelMax, CxGetJntAccel, CxGetAccelType, CxSetAccelType
**CxSetJntAccelMax**

Sets the maximum joint acceleration and deceleration values

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxSetJntAccelMax(CxMechanism mech, long jnt_number, double accel_rise_max, double accel_fall_max)
```

**ARGUMENTS**

- **mech** The mechanism ID of mechanism open for control
- **jnt_number** The independent joint number (count from 0)
- **accel_rise_max** The positive maximum joint acceleration (if it is a revolute joint, the unit is deg/sec^2 or rad/sec^2, depending on unit. The default unit is degrees. See CxSetUnit in the Nodes, Frames and Attributes section of the CODE API Programmers Reference Manual- Volume 2 for more information).
- **accel_fall_max** The positive maximum joint deceleration (if it is a revolute joint, the unit is deg/sec^2 or rad/sec^2, depending on unit. The default unit is degrees. See CxSetUnit in the Nodes, Frames and Attributes section of the CODE API Programmers Reference Manual- Volume 2 for more information).

**DESCRIPTION**

This function allows the user to set the maximum joint acceleration and deceleration values.

**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>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_JOINT_NOT_FOUND</td>
<td>The specified joint does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_JOINT_NUMBER</td>
<td>The Joint number entered is not valid.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>The mechanism is not opened for control.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The given node is cut out.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following example sets the maximum limits for acceleration and deceleration for a joint, based on the specified safety factor.

```c
#include <code/robpac.h>
```
#define TOT_JOINTS 6
#define MAX_ACCEL 0.8
#define MAX_DECEL 0.8

void main (void)
{
    CxServer Server;
    CxMechanism mech;
    CxNodeId robot;
    long jnt_nr;
    long high_safety_factor;

    Server = CxOpenServer( "Testing", CX_SMEM, 0 );
    CxGetNamedNodeId( Server, "robot", &robot );
    mech = CxOpenMechanism( Server, robot, CX_CONTROL );

    /* Set maximum robot acceleration and deceleration values */
    if (high_safety_factor)
        CxSetJntAccelMax( mech, jnt_nr, MAX_ACCEL*0.33, MAX_DECEL *0.33 );
    else
        CxSetJntAccelMax( mech, jnt_nr, MAX_ACCEL*0.67, MAX_DECEL *0.67 );

    CxRobpacExit();
}

WARNINGS
Since the allowable maximums are robot- and design-dependent, the user must be extremely cautious when using these values to actually control a robot. Excessive accelerations may cause the robot’s inertial loads to damage the drive transmissions and other mechanical components on the robot.

HARDWARE AND SYSTEM DEPENDENCIES
The number that is entered is the effective acceleration rate allowed, considering the joint transmission mechanisms and the effective inertia of all links which the motor must accelerate.

SEE ALSO
CxGetScrewAccelMax, CxGetJntAccelMax, CxSetJntAccel, CxGetJntAccel
**CxSetJntDepConst**

Sets the joint dependency parameters for a linearly dependent joint

**SYNOPSIS**
```
#include <code/robpac.h>
long CxSetJntDepConst(CxMechanism mech, long dep_jnt_number, long indep_jnt_number, double dconst)
```

**ARGUMENTS**
- **mech**
  - The mechanism ID of mechanism open for control
- **dep_jnt_number**
  - The linearly dependent joint number (starts from 0)
- **indep_jnt_number**
  - Joint number of the independent joint upon which the dependent joint depends (starts from 0)
- **dconst**
  - The joint's linear dependency constant

**DESCRIPTION**
This function is used to set the joint's linear dependency constant and the independent joint number upon which it depends (see CxGetJntDepConst). The linear dependency constant relates the dependent joint to the independent joint, as follows: Value of dependent joint: linear dependency of constant × value of independent joint.

**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>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_JOINT_NOT_FOUND</td>
<td>The specified joint does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_JOINT_NUMBER</td>
<td>The Joint number entered is not valid.</td>
</tr>
<tr>
<td>CX_JNT_NOT_LINEAR_DEP</td>
<td>The joint is not of type CX_LIN_DEP.</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>The mechanism is not opened for control.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The given node is cut out.</td>
</tr>
</tbody>
</table>

**EXAMPLE**
The following example opens a mechanism, then performs the following task for each joint within that mechanism. First, it gets the joint dependency type. If the current joint is of type CX_LIN_DEP, the code gets the dependency constant and the node number on which the current node depends. Next, it prints out this information. Finally, depending on the selected gear ratio, it resets the dependency ratio to either 0.5 or 3.0.

```
#include <stdio.h>
#include <code/robpac.h>
```
#define  TOT_JOINTS  6

void main (void)
{
    CxServer Server;
    CxMechanism mech;
    CxNodeId robot;
    long   dependency_type;
    double depend_const;
    long   jnt_nr;
    long   lower_gear_ratio;
    long   indep_jnt_nr;
    
    Server = CxOpenServer( "Testing", CX_SMEM, 0 );
    CxGetNamedNodeId( Server, "robot", &robot );
    
    mech = CxOpenMechanism( Server, robot, CX_CONTROL );
    for  (jnt_nr=0; jnt_nr<TOT_JOINTS; jnt_nr++) {
        CxGetJntDepType( mech, jnt_nr, &dependency_type );
        if ( dependency_type == CX_LIN_DEP )
        {
            CxGetJntDepConst( mech, jnt_nr, &indep_jnt_nr, &depend_const );
            printf(" Joint # %ld is Linearly dependent on Joint #\%ld\n", jnt_nr, indep_jnt_nr );
            printf(" Dependency factor is \%lf \n", depend_const );
            if (lower_gear_ratio)
                CxSetJntDepConst( mech, jnt_nr, indep_jnt_nr, 0.5 );
            else
                CxSetJntDepConst( mech, jnt_nr, indep_jnt_nr, 3.0 );
        }
    }
    
    CxRobpacExit();
}

SEE ALSO

CxGetJntDepConst, CxGetJntDepType, CxSetJntDepType
CxSetJntDepType
Sets the joint dependency type

SYNOPSIS
#include <code/robpac.h>
long CxSetJntDepType(CxMechanism mech, long jnt_number, long type)

ARGUMENTS
mech The mechanism ID of mechanism open for control
jnt_number The joint’s number (count from 0)
type The joint’s dependency type

DESCRIPTION
This function is used to set the joint’s dependency type for the given joint number. The default is CX_INDEPEND. The valid dependency types are:
- CX_INDEPEND
- CX_LIN_DEP
- CX_FUNCTION

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>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_JOINT_NOT_FOUND</td>
<td>The specified joint does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_JOINT_NUMBER</td>
<td>The Joint number entered is not valid.</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>The dependency type entered is not CX_INDEPEND, CX_LIN_DEP, or CX_FUNCTION.</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>The mechanism is not opened for control.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The given node is cut out.</td>
</tr>
</tbody>
</table>

EXAMPLE
The following example first adds a rotational joint to the robot. Next, it adds a second joint, then makes the second joint dependent on the first one, setting the dependency type to CX_LIN_DEP.

#include <code/robpac.h>
#define TOT_JOINTS 6
void main (void)
{
    CxServer    Server;
    CxMechanism mech;
    CxNodeId    robot, jnt1, jnt2;
    long        jnt1_nr, jnt2_nr;

    Server = CxOpenServer( "Testing", CX_SMEM, 0 );
    CxGetNamedNodeId( Server, "robot", &robot );
    mech = CxOpenMechanism( Server, robot, CX_CONTROL );

    CxAddNewNode ( Server, "Joint_1", robot );
    CxGetNamedNodeId( Server, "Joint_1", &jnt1 );
    CxAddJoint( Server, jnt1 );
    CxGetJntNumber( mech, jnt1, &jnt1_nr );

    CxAddNewNode ( Server, "Joint_2", robot );
    CxGetNamedNodeId( Server, "Joint_2", &jnt2 );
    CxAddJoint( Server, jnt2 );
    CxGetJntNumber( mech, jnt2, &jnt2_nr );
    CxSetJntDepType( mech, jnt2_nr, CX_LIN_DEP );
    CxSetJntDepConst( mech, jnt1_nr, jnt1_nr, 0.5 );
    .

SEE ALSO

    CxGetJntDepType, CxGetJntDepConst, CxSetJntDepConst
**CxSetJntInfinite**

Sets an independent rotational joint to infinite rotation

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxSetJntInfinite(CxMechanism mech, long jnt_number, long type)
```

**ARGUMENTS**

- `mech`: The mechanism ID or mechanisms open for control
- `jnt_number`: The independent joint number (starts from 0)
- `type`: The flag which determines whether joint is infinite (`CX_TRUE`) or not (`CX_FALSE`)

**DESCRIPTION**

This function is used to override the joint limits normally imposed for a rotational joint. The specified joint can then rotate indefinitely about the joint axis in either direction. If `CxMoveSingleAxis()` is used, the user can enter any value for the rotation amount and the joint will continue to rotate until reaching the specified value.

When using target frames and inverse kinematics to determine joint values, the values of the infinite joints will always be returned in the range from -180 to +180 degrees after the motion is completed; however, the motion will not be constrained to this range. For example, if a value is initially at +150 degrees and the inverse kinematics determines a target joint value of -150 degrees, then the robot will move by +60 degrees, because this is the minimum rotation. If the user is jogging using a teach pendant, the joint value will not be "rolled over" and will retain its jogged value.

**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>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_WRONG_TYPE_JOINT</td>
<td>The specified node is not a rotational joint.</td>
</tr>
<tr>
<td>CX_JOINT_NOT_FOUND</td>
<td>The specified joint does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_JOINT_NUMBER</td>
<td>The Joint number entered is not valid.</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>Type entered is not <code>CX_TRUE</code> or <code>CX_FALSE</code>.</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>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The given node is cut out.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following example adds a new node and makes it a joint node. The code then makes the joint node an infinite joint.
#include <code/robpac.h>

void main (void)
{
    CxServer Server;
    CxMechanism mech;
    CxNodeId robot, jnt6;
    long jnt6_nr;

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

    CxGetNamedNodeId( Server, "robot", &robot );

    mech = CxOpenMechanism( Server, robot, CX_CONTROL );

    CxAddNewNode( Server, "Joint_6", robot );
    CxGetNamedNodeId( Server, "Joint_6", &jnt6 );
    CxAddJoint( Server, jnt6 );
    CxGetJntNumber( mech, jnt6, &jnt6_nr );
    CxSetJntInfinite( mech, jnt6_nr, CX_TRUE );

    //
}

SEE ALSO

CxGetJntInfinite
CxSetJntType
Sets the joint type to rotational or translational

SYNOPSIS
#include <code/robpac.h>
long CxSetJntType(CxMechanism mech, long jnt_number, long type)

ARGUMENTS
mech       The Mechanism ID of mechanism open for control
jnt_number The joint number (count from 0)
type       The joint type (CX_ROTATE or CX_TRANS)

DESCRIPTION
This function is used to set the joint type (rotational or translational) for the given joint. Joints are rotational by default.

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>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_JOINT_NOT_FOUND</td>
<td>The specified joint does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_JOINT_NUMBER</td>
<td>The Joint number entered is not valid.</td>
</tr>
<tr>
<td>CX_INVALID_ARGUMENT</td>
<td>The type entered is not CX_ROTATE or CX_TRANS.</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPEN_FOR_CONTROL</td>
<td>The mechanism is not opened for control.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The given node is cut out.</td>
</tr>
</tbody>
</table>

EXAMPLE
The following example adds a new node and makes it a joint node. It then sets the joint type to translational.
#include <code/robpac.h>

void main (void)
{
    CxServer    Server;
    CxMechanism mech;
    CxNodeId    robot, jnt6;
    long        jnt6_nr;

    Server = CxOpenServer( "Testing", CX_SMEM, 0 );
CxGetNamedNodeId( Server, "robot", &robot );

mech = CxOpenMechanism( Server, robot, CX_CONTROL );
.
.
CxAddNewNode( Server, "Joint_6", robot );
CxGetNamedNodeId( Server, "Joint_6", &jnt6 );
CxAddJoint( Server, jnt6 );
CxGetJntNumber( mech, jnt6, &jnt6_nr );
CxSetJntType( mech, jnt6_nr, CX_TRANS );
.
.
}

**WARNING**

The user must be extremely cautious when changing the type of a joint. The user should never change the joint type at run-time.

**SEE ALSO**

CxGetJntType
**CxSetMaxDofValue**

Sets the maximum joint value (upper limit)

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxSetMaxDofValue(CxMechanism mech, long jnt_number, double dof)
```

**ARGUMENTS**

- `mech` The mechanism ID of mechanism open for control
- `jnt_number` The Joint number (count from 0)
- `dof` The maximum joint motion (if it is a revolute joint, units are in degrees or radians, depending on the unit. Default is degrees [defined in `<code/const.h>`]. See `set_unit` in the Nodes, Frames and Attributes section of the CODE API Programmers Reference Manual- Volume2 for more information).

**DESCRIPTION**

This function is used to set the maximum joint value.

**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>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_JOINT_NOT_FOUND</td>
<td>The specified joint does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_JOINT_NUMBER</td>
<td>The Joint number entered is not valid.</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPENED_FOR_CONTROL</td>
<td>The mechanism is not opened for control.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The given node is cut out.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

CxGetMaxDofValue, CxSetMinDofValue
CxSetMaxJntSpeed
Sets the maximum joint speed

SYNOPSIS
#include <code/robpac.h>
long CxSetMaxJntSpeed(CxMechanism mech, long jnt_number, double speed)

ARGUMENTS
mech The mechanism ID or mechanism open for control
jnt_number The independent joint number (count from 0)
speed For revolute joints, the units are deg/sec or rad/sec, with the default being deg/sec (see CxSetUnit in the Nodes, Frames and Attributes section of the CODE API Programmers Reference Manual- Volume 2 for more information). For translational joints, the speed is in user units/sec.

DESCRIPTION
This function is used to set the maximum joint speed.

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>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_JOINT_NOT_FOUND</td>
<td>The specified joint does not exist.</td>
</tr>
<tr>
<td>CX_INVALID_JOINT_NUMBER</td>
<td>The Joint number entered is not valid.</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPENED_FOR_CONTROL</td>
<td>The mechanism is not opened for control.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The given node is cut out.</td>
</tr>
</tbody>
</table>

EXAMPLE
The following example gets the maximum joint speed set for each joint. It then increases or decreases the joint speed limit based on a set of software flags.
#include <code/robpac.h>
#define TOT_JOINTS 6

void main (void)
{
  CxServer Server;
  CxMechanism mech;
  CxNodeId robot;
  long jnt_nr;
  double max_speed;
long run_slower, run_faster;

Server = CxOpenServer( "Testing", CX_SMEM, 0 );
CxGetNamedNodeId( Server, "robot", &robot );
mech = CxOpenMechanism( Server, robot, CX_CONTROL );

for (jnt_nr=0; jnt_nr<TOT_JOINTS; jnt_nr++) {
    CxGetMaxJntSpeed( mech, jnt_nr, &max_speed);
    if ( run_slower )
        CxSetMaxJntSpeed( mech, jnt_nr, max_speed*0.9 );
    else if (run_faster )
        CxSetMaxJntSpeed( mech, jnt_nr, max_speed*1.1 );
}

CxRobpacExit();

SEE ALSO
    CxGetMaxJntSpeed
**CxSetMinDofValue**

Sets minimum joint value (lower limit)

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxSetMinDofValue(CxMechanism mech, long jnt_number, double dof)
```

**ARGUMENTS**

- `mech`  
  The mechanism ID of mechanism open for control

- `jnt_number`  
  The Joint number (count from 0)

- `dof`  
  The lower limit of a joint value. For revolute joints, the units can be either degrees or radians, the default being degrees (see `CxSetUnit` in the Nodes, Frames and Attributes section of the CODE API Programmers Reference Manual - Volume 2 for more information). For translational joints, the value is expressed in user units.

**DESCRIPTION**

This function is used to set the minimum joint limit.

**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>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_JOINT_NOT_FOUND</td>
<td>The specified joint does not exist.</td>
</tr>
<tr>
<td>CX_NO_ROB_FOR_THE_JOINT</td>
<td>An invalid mechanism ID, or the mechanism does not have a robot assigned to it.</td>
</tr>
<tr>
<td>CX_INVALJOINTNUMBER</td>
<td>The Joint number entered is not valid.</td>
</tr>
<tr>
<td>CX_INVALID_MECHANISM</td>
<td>An invalid mechanism ID, or the mechanism does not exist.</td>
</tr>
<tr>
<td>CX_MECH_NOT_OPENED_FOR_CONTROL</td>
<td>The mechanism is not opened for control.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The given node is cut out.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

CxGetMinDofValue, CxGetMaxDofValue, CxSetMaxDofValue
Geometry

The Geometry and Pose API functions modify the workcell database file (also called the CIMServer Cell Model) currently loaded by the CIMServer. The API functions work with all versions of the CIMServer—in other words, you can use these API functions to change the geometry or pose of any node in the loaded workcell regardless of whether you are using the CIMulation or CIMControl version of the CIMServer.

NOTE: Workcell nodes are also referred to as tree nodes or elements in this document.

NOTE: The words frame and segment will be used interchangeably to describe a segment consisting of the initial frame and the segment interpolation type (linear, circular, etc., see below). In actuality, the frame of the next segment is also required to complete a full segment description, but since the curve is made up of a linked list of frames and interpolation typed, the next required frame is obtained from the data structure of the next segment, thus eliminating redundancy.
**CxAddCurveSeg**

Adds a curve segment to the set of linked curve segments

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxAddCurveSeg(CxServer Server, CxNodeId node, char *seg_name, char *new_seg_name, char axes[4], double a1, double a2, double a3, double x, double y, double z)
```

**ARGUMENTS**

- **Server**
  - The Server ID
- **node**
  - The curve node ID
- **seg_name**
  - The name of existing segment
- **new_seg_name**
  - The name of new segment to insert after the seg_name
- **axes**
  - The order of rotation for angles (e.g. "XYZ") of segment frame relative to the curve node (not segment) frame
- **a1, a2, a3**
  - The relative rotation angles. The angles are in degrees unless the unit is set to CX_USE_RADIAN. See CxSetUnit in the Nodes, Frames and Attributes section of the CODE API Programmers Reference Manual- Volume 2 for more information.
- **x, y, z**
  - The relative position components relative to curve node (not segment) frame

**DESCRIPTION**

This function allows the user to add (link) a named curve segment immediately after an existing segment. If the existing segment is not named, but entered as "" (the empty string), then the new segment is inserted as the first curve segment in the sequence. The new segment defaults to a linear type curve segment, unless it is the terminal frame, in which case the curve segment type is terminal. The node requires the GEOMETRY attribute.

**NOTE:** If the linear curve type is not desired, functions such as CxSetCurveSegArc can be used to change the segment’s type.

**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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_MISMATCHED_GEMOT_TYPE</td>
<td>The node is not of type CX_CURVE or CX_SWEPTT or CX_SWEPTR or CX_TRIAD.</td>
</tr>
<tr>
<td>CX_CURVE_SEG_NOT_NAMED</td>
<td>The new_seg_name is not valid.</td>
</tr>
<tr>
<td>CX_NO_SUCH_CURVE_SEG_NAME</td>
<td>The seg_name is not in the curve linked list.</td>
</tr>
</tbody>
</table>
### Error Codes

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The node does not have GEOMETRY attribute.</td>
</tr>
<tr>
<td>CX_REDUNDANT_CURVE_SEG_NAME</td>
<td>A curve segment of that name already exists.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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>

### WARNING

Users can build a curvilinear path using this function, by adding segments relative to any existing segment in a previously defined curve. However, the user must have already generated a curve node either through the CIMTools interface or by using the API functions `CxAddNewNode` (See the Nodes, Frames and Attributes section of the CODE API Programmer’s Reference Manual - Volume 2), and `CxSetRelPose`.

### EXAMPLE

The following example is a complete CODE application process. When compiled and linked properly, it will run with the CIMServer in simulation, and accomplish the tasks described below.

The code makes a path on top of an existing table by adding 6 curve segments (named pt1 through pt6). The `CxGetCurveSegName` command is then used to ensure that the segments were added. Each segment of the path is then given a curve segment type. All curve segment types (`CX_LINEAR`, `CX_CIRCULAR`, `CX_LINCIRC`) are used in the path. The path is then traced by the robot using the `CxMoveRelPath` command. When the path is traced completely, the robot moves away from the table and the curve segments are deleted.

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

void main(void)
{
    CxServer Server;
    CxMechanism mech;
    CxNodeId RT5000, tcf, path;
    double  tol, axis[4];
    char    seg[CX_MAXNAME];

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

    CxGetNamedNodeId( Server, "RT5000", &RT5000 );
    CxGetNamedNodeId( Server, "path", &path );
    CxGetNamedNodeId( Server, "tcf", &tcf );

    mech = CxOpenMechanism( Server, RT5000,CX_CONTROL );
    CxErrorPolicy(CX_RETURN_ERRORS);
    CxViewSwitch( Server, path, 1 );
    CxChangeColor(Server, path, 0.5, 0., 1.);
    CxFrameSwitch(Server, path, 0);

    /* add the six curve segments */
    CxAddCurveSeg( Server, path, "", "pt1", "ZYX", 270., 0., 0.,
                  0., 0., 0.);
    CxAddCurveSeg( Server, path, "pt1", "pt2", "ZYX", 0., 0., 0.,
                  0., 0., 0.);
```
250., 0., 0.);
CxAddCurveSeg( Server, path, "pt2", "pt3", "ZYX", 90., 0., 0.,
250., 450., 0.);
CxAddCurveSeg( Server, path, "pt3", "pt4", "ZYX", 270., 0., 0.,
0., 375., 0.);
CxAddCurveSeg( Server, path, "pt4", "pt5", "ZYX", 0., 0., 0.,
0., 100., 0.);
CxAddCurveSeg( Server, path, "pt5", "pt6", "ZYX", 0., 0., 0.,
0., 0., 0.);

/* get names of curve segments using both previous and next flags */
CxGetCurveSegName( Server, path, "pt2", seg, 1);
printf("%s  ", seg);
CxGetCurveSegName( Server, path, "pt1", seg, 0);
printf("%s \n", seg);

/* give each segment a curve type (default is linear) */
CxSetCurveSegArc3( Server, path, "pt1", 100., -75., 0.);
CxSetCurveSegLincirc( Server, path, "pt3", 1 );
CxSetCurveSegArc( Server, path, "pt4", 1 );
CxSetCurveSegLinear( Server, path, "pt2" );

/* move along the path using CxMoveRelPath and a move_thru
blend policy */
CxSetSimrate( Server, 0.4);
CxSetBlendPolicy( mech, CX_MOVE_THRU);
CxMoveRelPath( mech, path, tcf, "pt1", "pt6", "XYZ", 0., 0.,
0., 0., 0., 0.);
CxWaitForEndOfMotion( mech );

/* move the robot away from the curve */
axis[0] = 0.0;  axis[1] = 100.0; axis[2] = 570.0; axis[3] = 0.0;
CxMoveAllAxes( mech, axis );
CxWaitForEndOfMotion( mech );

/* delete curve segments pt1 through pt6 */
CxDeleteCurveSeg( Server, path, "pt1", "pt6" );
CxRobpacExit();

SEE ALSO

CxDeleteCurveSeg, CxSetCurveSegLincirc, CxSetCurveSegArc, CxSetCurveSegArc3,
CxSetCurveSegLinear
CxDecompCurveSeg
Decomposes curve segments into child segments

SYNOPSIS
#include <code/robpac.h>
long CxDecompCurveSeg(CxServer Server, CxNodeId node, char *seg_name,
char *final_seg_name, long tool_motion_type, long decomp_type,
long res_flag, long res, double tol)

ARGUMENTS
Server The Server ID
node The curve node ID
seg_name The name of the first segment to be decomposed
final_seg_name The name of the final segment to be decomposed
tool_motion_type The motion type: either CX_FULL_POSE or CX_Z_POSE
decomp_type The decomposition type: CX_LINEAR_INTERP or
CX_CIRCULAR_INTERP for decomposing higher order curves
res_flag The flag which determines whether resolution (CX_ON or 1), or tolerance
(CX_OFF or 0) is used for decomposition
res The number of decomposed children for the segments
tol The fit accuracy of the child segments to the curve segment

DESCRIPTION
This function allows the user to decompose one or more curve segments into smaller segments. This is useful
for curves that will be swept to form a polyhedron by extruding or revolving. Types CX_LINEAR_INTERP,
CX_LINCIRC_INTERP, and CX_CIRCULAR_INTERP will decompose to the same child types. Higher
order types (conic, cubic, B-Splines, etc.) will decompose to linear and/or circular types, depending on
whether the decomp_type is specified as CX_LINEAR_INTERP or CX_CIRCULAR_INTERP.

The intermediate child segment frames will be interpolated (posed) proportional to the distance along the
curve segment. If CX_FULL_POSE is specified, a general screw angle formulation is used to reorient the
child segment frame as a proportional rotation about the screw axis. A six axis robot may use this mode to
orient a tool frame using general screw formulations.

If CX_Z_POSE is specified, the intermediate child frames are posed so that their Z axis is proportionally
oriented between the initial segment frame orientation and the next segment frame orientation. The procedure
defines an axis normal to the plane defined by the cross product of the initial segment Z axis and the next
segment Z axis, then rotates the initial segment about this axis proportionally to the child segment distance.
This interpolation procedure will allow robots of fewer joints (and less orientation capability, such as machine
tools) to move along paths which are decomposed into child segments.

If either seg_name or final_seg_name is entered, each segment between seg_name and
final_seg_name, and including final_seg_name, will be decomposed into child segments according
to res_flag, res, and tol settings. If both segments are specified by " "(the empty string), then the
algorithm will decompose the longest non-linear segment according to res_flag, res, and tol, and use
its child segment length to decompose the other segments. In this way, the child segments of all segments are
approximately the same length.

If seg_name is " ", then all segments from the first segment to and including final_seg_name are
decomposed. If final_seg_name is " ", then all segments from and including seg_name to and
including the last segment are decomposed. The curve node ID requires 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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_NO_SUCH_CURVE_SEG_NAME</td>
<td>The given node is not of type CX_CURVE or seg_name or final_seg_name are not found in linked list of curve segments.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The node does not have GEOMETRY attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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_MISMATCHED_GEOM_TYPE</td>
<td>The node is not of type CX_CURVE or CX_SWEPTT or CX_SWEPTR.</td>
</tr>
</tbody>
</table>

WARNING

Users can modify a curvilinear path using this function but the action is non-recoverable, unless other APIs are used to save and restore the segments.

EXAMPLE

The following example gives each segment a curve type and then sets the resolution of the segments to give them a smoother look.

```c
CxServer Server
CxNodeId;

CxSetCurveSegArc3( Server, path, "pt1", 100., -75., 0. );
CxSetCurveSegLincirc( Server, path, "pt3", 1 );
CxSetCurveSegArc( Server, path, "pt4", 1 );
CxSetCurveSegLinear( Server, path, "pt2" );

/* Give the curves resolution of 8 */
CxDecompCurveSeg( Server, path, "", "", CX_FULL_POSE, CX_CIRCULAR_INTERP, 1, 8, tol);
```

SEE ALSO

CxGetCurveSegName, CxAddCurveSeg
CxDeleteCurveSeg

Deletes a curve segment from a set of linked curve segments

SYNOPSIS

```c
#include <code/robpac.h>
long CxDeleteCurveSeg(CxServer Server, CxNodeId node, char *seg_name,
                     char *final_seg_name)
```

ARGUMENTS

- **Server**: The Server ID
- **node**: The curve node ID
- **seg_name**: The name of the first segment to be deleted
- **final_seg_name**: The name of the last segment to be deleted

DESCRIPTION

This function allows the user to delete one or more segments from a sequence of segments in a curve. If `seg_name` is entered as "" (the empty string), then the segment to be deleted will be the first curve segment. If the `final_seg_name` is also "", then all curve segments except the original node with the curve geometry are deleted. If `seg_name` is not "", but `final_seg_name` is "", then all segments from `seg_name` to the last segment are deleted. The segment previous to `seg_name` then becomes the terminal segment. If `seg_name` and `final_seg_name` are not " " and entered as the same name, then a single segment is deleted (e.g., `seg_name` = "f3" and `final_seg_name` = "f3"). The curve node ID requires 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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_NO_SUCH_CURVE_SEG_NAME</td>
<td>The given node is not of type CX_CURVE or no segment of given name(s) exists.</td>
</tr>
<tr>
<td>CX_NO_CURVE_SEG_TO_DELETE</td>
<td>No curve segments to delete.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The node does not have the GEOMETRY attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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_MISMATCHED_GEOM_TYPE</td>
<td>The node is not of type CX_CURVE or CX_SWEPTT or CX_SWEPTR.</td>
</tr>
</tbody>
</table>
EXAMPLE

The following example adds six curve segments then deletes them:

```c
CxServer Server;
CxNodeId path;

/* Add the six curve segments */
CxAddCurveSeg( Server, path, "", "pt1", "ZYX", 270., 0., 0., 0., 0., 0.);
CxAddCurveSeg( Server, path, "pt1", "pt2", "ZYX", 0., 0., 0., 250., 0., 0.);
CxAddCurveSeg( Server, path, "pt2", "pt3", "ZYX", 90., 0., 0., 250., 450., 0.);
CxAddCurveSeg( Server, path, "pt3", "pt4", "ZYX", 270., 0., 0., 0., 375., 0.);
CxAddCurveSeg( Server, path, "pt4", "pt5", "ZYX", 0., 0., 0., 0., 100., 0.);
CxAddCurveSeg( Server, path, "pt5", "pt6", "ZYX", 0., 0., 0., 0., 0., 0.);

/* Delete curve segments pt1 through pt6 */
CxDeleteCurveSeg( Server, path, "pt1", "pt6" );
```

SEE ALSO

CxAddCurveSeg
CxGetBoolean

Gets the Boolean parts and operation for a given Boolean result

SYNOPSIS

#include <code/robpac.h>
long CxGetBoolean(CxServer Server, CxNodeId node, CxNodeId *node_a,
    long *operation, CxNodeId *node_b)

ARGUMENTS

Server        The Server ID
Node          The Boolean result node
node_a        The Boolean component A node
Operation     The Boolean operation assigned: CX_SUB, CX_ADD, or CX_INTRX
    CX_SUB          The difference between the two solids
    CX_ADD          The union of the two solids
    CX_INTRX       The intersection of the two Boolean components
node_b        The Boolean component B node

DESCRIPTION

This function is used to get the Boolean components and operation for a given Boolean result node. If the
Boolean components A and/or B are deleted, the corresponding node ID (number and index) will contain
zero. The Boolean result node requires 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>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 given Boolean result node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given Boolean result node does not have the GEOMETRY attribute.</td>
</tr>
<tr>
<td>CX_BOOLEAN_NOT_FOUND</td>
<td>The given Boolean result node is not of the BOOLEAN type.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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 example is a complete CODE application process. When compiled and linked properly, the code will run
with the CIMServer, and accomplish the desired task described below.

This process first makes a node named box as a child of the world, then adds another node named cyl as
child of the world also. The cyl node is posed at the center of the box, using the function
CxSetRelPose. The Boolean resulting node bool is then added with the GEOMETRY attribute and CxMakeBoolean is used to make the geometry as the result of subtracting cyl from box, or bool = box - cyl.

Finally, the process uses CxGetBoolean to get the boolean parts of the Boolean result bool just made, and prints the result to a standard output.

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

void main(void)
{
    CxServer Server;
    CxNodeId world, box, cyl, bool, ele_a, ele_b;
    CxVector ang, vec;
    long 1;
    double tol = 0.2;
    long res = 12;
    long operation;
    char op[6], name_a[CX_MAXNAME], name_b[CX_MAXNAME];

    /* connect to server named sample */
    Server = CxOpenServer("Sample", CX_SMEM, 0);

    /* add new node named box as a child of world */
    CxGetNamedNodeId ( Server, "world", &world );
    CxAddNewNode( Server,"box", world );
    CxGetNamedNodeId ( Server, "box", &box );
    CxAddGeometry( Server, box );
    CxMakeBox( Server, box, 100., 100., 100. );

    /* add new node named cyl as a child of world */
    CxAddNewNode( Server, "cyl", world );
    CxGetNamedNodeId ( Server, "cyl", &cyl );
    CxAddGeometry( Server, cyl );

    /* position the cyl at the center of the box */
    for(i=0; i<3; i++) {
        ang[i] = 0.0;
    }
    vec[0] = 50.;
    vec[1] = 50.;
    vec[2] = -50.;
    CxSetRelPose( Server, cyl, "XYZ", ang, vec );
    CxMakeCyl( Server, cyl, 30., 200., 1, &res, &tol );

    /* add new node named bool as a child of world */
    CxAddNewNode( Server, "bool", world );
    CxGetNamedNodeId ( Server, "bool", &bool );
    CxAddGeometry( Server, bool );

    /* make bool as the result of box subtract cyl */
    CxMakeBoolean( Server, bool, box, CX_SUB, cyl );

    /* get Boolean components for bool */
}```
if(CxGetBoolean( Server, bool, &ele_a, &operation, &ele_b) == CX_ERROR ) {
    printf(" CX_ERROR: CxGetBoolean() API failed...
");
} else {
    if(operation == CX_SUB) {
        strcpy(op, "CX_SUB");
    } else if(operation == CX_ADD) {
        strcpy(op, "CX_ADD");
    } else {
        strcpy(op, "CX_INTRX");
    }
    CxGetNodeName ( Server, ele_a, name_a );
    CxGetNodeName ( Server, ele_b, name_b );
    printf("\n bool = %s %s %s
", name_a, op, name_b);
}

/* exit */
CxRobpacExit();

SEE ALSO

CxMakeBoolean
CxGetBox

Gets the dimensions of a given box from the CIMServer workcell

SYNOPSIS

```c
#include <code/robpac.h>
long CxGetBox(CxServer Server, CxNodeId node, double *L1, double *L2, double *L3)
```

ARGUMENTS

- **Server** The Sever ID
- **node** The node ID
- **L1** The box dimension in local x direction
- **L2** The box dimension in local y direction
- **L3** The box dimension in local z direction

DESCRIPTION

This function is used to get the dimensions of a given box from the CIMServer workcell. The node requires 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>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_MISMATCHED_GEOM_TYPE</td>
<td>The given element is not of type CX_BOX.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The node does not have GEOMETRY attribute.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>The given 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_INVALID_NODE_ID</td>
<td>The node number does not match the given The node ID number.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following example is a complete CODE application process. This CODE application process first makes a new tree node named base_box as a child of world. The process then adds the geometry attribute to the node, and makes a box of dimension (100., 200., 300.). The function CxGetBox is then used to demonstrate how to get the dimensions of a box in the workcell model. Finally, the box dimensions are modified, and the base_box is remade with new dimensions.

```c
#include <stdio.h>
#include <code/robpac.h>
void main(void)
```
{  
    CxServer Server;
    double L1, L2, L3;
    CxNodeId world, box;

    /* connect to server named Sample */
    Server = CxOpenServer("Sample", CX_SMEM, 0);

    CxGetNamedNodeId ( Server, "world", &world );

    /* add a node named base_box as a child of world */
    CxAddNewNode( Server, "base_box", world );
    CxGetNamedNodeId ( Server, "base_box", &box );

    /* add geometry attribute to node base_box */
    CxAddGeometry( Server, box );

    /* make a box of the given dimensions */
    if( CxMakeBox( Server, box, 100., 200., 300.) == CX_ERROR ) {
        fprintf(stderr,"CX_ERROR making base_box...
");  
    }

    /* get base_box dimensions */
    CxGetBox( Server, box, &L1, &L2, &L3);

    /* modify base_box dimensions */
    L1 += 10.;
    L2 += 20.;
    L3 += 30.;

    /* remake base_box with modified dimensions */
    CxMakeBox( Server, box, L1, L2, L3);

    /* exit */
    CxRobpacExit();
}

SEE ALSO

CxMakeBox
**CxGetCone**

Gets the dimensions of a given cone from the CIMServer workcell

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxGetCone(CxServer Server, CxNodeId node, double *radius, double *length, long *res_flag, long *res, double *tol)
```

**ARGUMENTS**

- **Server** The Server ID
- **node** The node ID
- **radius** The cone radius
- **length** The cone length
- **res_flag** Shows whether res or tol is used. When set to CX_ON (1), res is used to determine tol. When set to CX_OFF (0), tol is used to determine res.
- **tol** The cone tolerance, the error in the faceted approximation to a true cone

**DESCRIPTION**

This function gets the dimensions of the given cone from the CIMServer workcell. The node requires the `GEOMETRY` attribute.

Polygons are used to approximate the cone surfaces. The higher the resolution or the lower the tolerance, the closer the approximation is to a true cone. Resolution and tolerance are dependent. Once a resolution is given for a cone with known radius, tolerance is calculated, and vice versa.

The following figure depicts the relationship between resolution, tolerance, and cone radius.

![Diagram of cone with resolution (res) and tolerance (tol) annotations](image)

**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>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_MISMATCHED_GEOM_TYPE</td>
<td>The given element is not of type CONE.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The node does not have GEOMETRY attribute.</td>
</tr>
<tr>
<td>CX_NODE_NOT_FOUND</td>
<td>The given 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>Error Codes</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------------------------------------------------</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 uses `CxGetCone` to find the resolution of the cone named `cone`. It then makes sure that the cone has a resolution greater than or equal to 8.

```c
CxServer Server;
CxNodeId cone;
double r, len, tol;
long res_flg, res;

/* get cone dimensions from the cell model */
CxGetCone( Server, cone, &r, &len, &res_flg, &res, &tol );

/* check its resolution */
if(res < 8) {
    res = 8;
    /* make new cone with resolution 8 */
    CxMakeCone( Server, cone, r, len, CX_ON, &res, &tol);
}
```

**SEE ALSO**

`CxMakeCone`
CxGetCurveSegFrame

Gets the frame matrix for a segment in a curve segment

SYNOPSIS

```c
#include <code/robpac.h>
CxGetCurveSegFrame(CxServer Server, CxNodeId node, char *seg_name, char axes[4], CxVector angles, CxVector vec)
```

ARGUMENTS

- **Server**: The Server ID
- **node**: The curve node ID
- **seg_name**: The name of the segment
- **axes**: The principal body axes about which the frame is rotated and the order of rotation (e.g. ZYX)
- **angles**: The relative rotation angles in the order that the axes were returned
- **vec**: The relative position vector as XYZ components.

NOTE: Type `CxVector` is defined in `<code/matx_defs.h>` as follows:
```c
typedef double CxVector[3];
```

DESCRIPTION

This function returns the principal axes about which the frame is rotated and the order of rotation which poses a segment of name `seg_name` relative to the curve node. If the name is entered as " " (the empty string), then the frame of the first segment will be returned; otherwise, the frame pose vectors of the first segment with name `seg_name` will be returned. Since swept elements of type `CX_SWEPTT` or `CX_SWEPTR` also have the `CX_CURVE` attribute, these can be edited as a curve. The curve node requires 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>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 given curve node does not exist.</td>
</tr>
<tr>
<td>CX_NO_SUCH_CURVE_SEG_NAME</td>
<td>The <code>seg_name</code> does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The node does not have the <code>GEOMETRY</code> attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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_MISMATCHED_GEOM_TYPE</td>
<td>The node is not of type <code>CX_CURVE</code> or <code>CX_SWEPTT</code> or <code>CX_SWEPTR</code>.</td>
</tr>
</tbody>
</table>
EXAMPLE

The following example gets the pose vector for a segment named f4. It then loads the origin of the segment frame into the local variables X, Y, and Z.

```c
CxServer Server;
CxNodeId curve;
char axes[4];
CxVector angles;
CxVector vec;
double X, Y, Z;

CxGetCurveSegFrame(Server, curve, "f4", axes, angles, vec);

X = vec[0];
Y = vec[1];
Z = vec[2];
```

SEE ALSO

`CxGetCurveSegName`
CxGetCurveSegName

Gets the previous or next segment name given a reference seg name

SYNOPSIS

```c
#include <code/robpac.h>
long CxGetCurveSegName( CxServer Server, CxNodeId node,
                        char *ref_seg_name, char *seg_name, long prev_or_next)
```

ARGUMENTS

- **Server**: The Server ID
- **node**: The curve node ID
- **ref_seg_name**: The name of the reference segment
- **seg_name**: The returned name of the previous or the next segment
- **prev_or_next**: The flag which determines whether to use the previous segment (CX_TRUE[1]), or the next segment (CX_FALSE[0])

DESCRIPTION

This function allows the user to get the name of a segment in a curve by using a reference segment frame. If the name of the segment previous to the reference segment is desired, the value for `prev_or_next` should be CX_TRUE. If the name of the segment following the reference segment is desired, `prev_or_next` should be CX_FALSE. The curve node requires the GEOMETRY attribute. See the following figure for more information. NOTE: If the reference segment name is entered as " " (the empty string) and `prev_or_next` is CX_TRUE, the first curve segment name will be returned. If the reference segment name is entered as " " (the empty string) and `prev_or_next` is CX_FALSE, the last curve segment name will be returned.

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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_NO_SUCH_CURVE_SEG_NAME</td>
<td>The reference name is not in the linked list of curve segments.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The node does not have the GEOMETRY attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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_MISMATCHED_GEOM_TYPE</td>
<td>The node is not of type CX_CURVE, CX_SWEPTT, or CX_SWEPTR.</td>
</tr>
</tbody>
</table>
WARNING

Users must provide a unique name for each segment in the linked list.

EXAMPLE

The following example adds six curve segments to the curve and then gets the names of the curve segments:

```c
CxServer Server;
char seg [CX_MAXNAME];
CxNodeId path;

/* Add the six curve segments */
CxAddCurveSeg( Server, path, "", "pt1", "ZYX", 270., 0., 0., 0., 0., 0.);
CxAddCurveSeg( Server, path, "pt1", "pt2", "ZYX", 0., 0., 0., 250., 0., 0.);
CxAddCurveSeg( Server, path, "pt2", "pt3", "ZYX", 90., 0., 0., 250., 450., 0.);
CxAddCurveSeg( Server, path, "pt3", "pt4", "ZYX", 270., 0., 0., 0., 375., 0.);
CxAddCurveSeg( Server, path, "pt4", "pt5", "ZYX", 0., 0., 0., 0., 100., 0.);
CxAddCurveSeg( Server, path, "pt5", "pt6", "ZYX", 0., 0., 0., 0., 0., 0.);

/* Get names of the first and last curve segments using both previous and next flags */
CxGetCurveSegName( Server, path, "", seg, CX_TRUE);
printf("first segment name: %s ", seg);
CxGetCurveSegName( Server, path, "", seg, CX_FALSE);
printf("last segment name: %s 
", seg);
```

SEE ALSO

CxAddCurveSeg, CxSetCurveSegName
CxGetCurveSegRestol

Gets the resolution parameters for a curve segment

SYNOPSIS

```
#include <code/robpac.h>
CxGetCurveSegRestol(CxServer Server, CxNodeId node, char *seg_name,
                    long *res_flag, long *res, double *tol)
```

ARGUMENTS

- **Server**: The server ID
- **node**: The curve node ID
- **seg_name**: The name of the segment
- **res_flag**: Shows whether res or tol is used. When set to CX_ON (1), res is used to determine tol. When set to CX_OFF (0), tol is used to determine res.
- **res**: The resolution which sets the number of faceted edges
- **tol**: The tolerance which sets the error in a faceted representation

DESCRIPTION

This function returns the res_flag, res, and tol settings for the named segment. The curve node requires 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>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 given curve node does not exist.</td>
</tr>
<tr>
<td>CX_MISMATCHED_GEOM_TYPE</td>
<td>The node is not of type CX_CURVE, CX_SWEPTT, or</td>
</tr>
<tr>
<td></td>
<td>CX_SWEPR.</td>
</tr>
<tr>
<td>CX_NO_SUCH_CURVE_SEG_NAME</td>
<td>The seg_name 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_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
<tr>
<td>CX_CURVE_SEG_NOT_NAMED</td>
<td>The seg_name is not a valid name.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The node does not have the GEOMETRY attribute.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following example first adds a curve segment to the path. Then it sets its type to CX_CIRCULAR_INTERP. Then it reads in the setting of tolerance/ resolution value for this segment. If either resolution is less than 5, or tolerance is greater than 50.0, it prints out an appropriate message.

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

void main(void)
{
    CxServer Server;
    CxNodeId path;
    Double tolerance;
    Long nr_subsegs, is_res;
    
    /* Add a new circular segment to the path */
    CxAddCurveSeg( Server, path, "pt2", "pt3", "ZXY",
                   0.0, 0.0, 0.0, 250.0, 0.0, 0.0);
    CxSetCurveSegArc( Server, path, "pt3", 0 );

    /* Get the current setting for tolerance and resolution */
    CxGetCurveSegRestol( Server, path, "pt3", &is_res, &nr_subsegs, &tolerance );
    if ( ((set_what == CX_ON) && (nr_subsegs<5)) ||
         ((set_what == CX_OFF) && (tolerance>50.0)) )
    {
        printf(" Resolution/Tolerance is inadequate for this \n object\n");
    }

    
    
SEE ALSO

    CxGetCurveSegName
**CxGetCurveSegType**

Gets the type of segment and the path axis of a curve segment

**SYNOPSIS**

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

**ARGUMENTS**

- **Server**
  - The Server ID
- **node**
  - The curve node ID
- **seg_name**
  - The name of the segment
- **type**
  - The segment type `CX_LINEAR_INTERP`, `CX_LINCIRC_INTERP`, `CX_CIRCULAR_INTERP`, or `CX_TERMINAL_FRAME`
- **path_axis**
  - If the segment type is `CX_LINCIRC_INTERP` or `CX_CIRCULAR_INTERP`, then `path_axis` should return value ranges from -1 to 2. See the description below.

**DESCRIPTION**

This function returns the type of segment for segment `seg_name`. Currently, the types returned are `CX_LINEAR_INTERP`, `CX_LINCIRC_INTERP`, `CX_CIRCULAR_INTERP`, or `CX_TERMINAL_FRAME`. Other types such as conic, cubic, etc., are decomposed into these more fundamental types for actual motion control. If the segment name is entered as " " (i.e. the empty string), then the type of the first segment will be returned; otherwise, the type of the first segment with name `seg_name` will be returned. The curve node requires the `GEOMETRY` attribute.

For types `CX_LINCIRC_INTERP` and `CX_CIRCULAR_INTERP`, `path_axis` will return a value 0, 1, or 2 corresponding to the x, y, or z axis as being the path tangent axis. The `CX_CIRCULAR_INTERP` type also may return a value of –1. This means three points are used to describe a circular arc, rather than two points and a tangent axis direction.

**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>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 given curve node does not exist.</td>
</tr>
<tr>
<td>CX_MISMATCHED_GEOM_TYPE</td>
<td>The node is not of type CX_CURVE or CX_SWEPT or CX_SWEPTR.</td>
</tr>
<tr>
<td>CX_NO_SUCH_CURVE_SEG_NAME</td>
<td>The <code>seg_name</code> does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The node does not have the <code>GEOMETRY</code> attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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 first adds a curve segment to the path. It then checks its type. If the segment is not the terminal segment and if its type is CX_LINEAR_INTERP, it sets the segment type to be CX_CIRCULAR_INTERP.

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

void main(void)
{
    CxServer Server;
    CxNodeId path;
    long seg_type, path_axis;

    /* Add new segment to the path */
    CxAddCurveSeg( Server, path, "pt2", "pt3", "ZXY", 0.0, 0.0, 0.0, 250.0, 0.0, 0.0);

    /* Set path type to be circular if appropriate */
    CxGetCurveSegType( Server, path, "pt3", &seg_type, &path_axis );
    if ( (seg_type != CX_TERMINAL_FRAME) && (seg_type == CX_LINEAR_INTERP) )
        CxSetCurveSegArc( Server, path, "pt3", 0);
}
```

SEE ALSO

CxSetCurveSegLinear, CxSetCurveSegLincirc
**CxGetCyl**

Gets the dimensions of a given cylinder from the CIMServer workcell

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxGetCyl(CxServer Server, CxNodeId node, double *radius, double *length, long *res_flag, long *res, double *tol)
```

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>The server ID</td>
</tr>
<tr>
<td>node</td>
<td>The node ID</td>
</tr>
<tr>
<td>radius</td>
<td>The cylinder radius</td>
</tr>
<tr>
<td>length</td>
<td>The cylinder length</td>
</tr>
<tr>
<td>res_flag</td>
<td>Shows whether res or tol was used. Whe set to CX_ON (1), res is used to determine tol. When set to CX_OFF (0), tol is used to determine res.</td>
</tr>
<tr>
<td>res</td>
<td>The cylinder resolution; the number of edges (and side faces) in a faceted approximation to a true cylinder</td>
</tr>
<tr>
<td>tol</td>
<td>The cylinder tolerance; error in the faceted approximation to a cylinder</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

This function is used to get the dimensions of the given cylinder from the CIMServer workcell. The node requires the **GEOMETRY** attribute.

Polygons are used to approximate the cylinder surfaces. The higher the resolution or the lower the tolerance, the closer the approximation is to a true cylinder. Resolution and tolerance are interdependent. Once a resolution is given for a cylinder with a known radius, tolerance is calculated, and vice versa. See CxGetCone for more information about the relationship between resolution, tolerance, and radius.

**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>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_ATTRIBUTE_NOT_FOUND</td>
<td>The node does not have the <strong>GEOMETRY</strong> attribute.</td>
</tr>
<tr>
<td>CX_MISMATCHED_GEOM_TYPE</td>
<td>The given element is not of type <strong>CYL</strong> (cylinder).</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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>The given node does not exist.</td>
</tr>
</tbody>
</table>
EXAMPLE

The following example uses CxGetCyl to find out the resolution of a cylinder named cyl, then uses CxMakeCyl (in this library) to ensure that the resolution of cyl is 4. This example will make the cylinder element look like a box on the screen.

```c
CxServer Server;
CxNodeId cyl;
double r,len,tol;
long flg, res;

/* get cylinder dimensions */
CxGetCyl( Server, cyl, &r, &len, &flg, &res, &tol );

/* change the cylinder resolution to 4 */
if (res != 4 ) {
    res = 4;
    CxMakeCyl( Server, cyl, r, len, CX_ON, &res, &tol );
}

SEE ALSO

CxMakeCyl,CxGetCone
```
CxGetFrust

Gets the dimensions of a given frustum from the CIMServer workcell

SYNOPSIS

```c
#include <code/robpac.h>
long CxGetFrust(CxServer Server, CxNodeId node, double *R1, double *R2,
                double *L, long *res_flag, long *res, double *tol)
```

ARGUMENTS

- **Server**  The Server ID
- **node**  The node ID
- **R1**  The frustum upper radius
- **R2**  The frustum lower radius
- **L**  The frustum length
- **res_flag**  Shows whether res or tol is used. When set to CX_ON (1), res is used to determine tol. When set to CX_OFF (0), tol is used to determine res.
- **res**  The frustum resolution; the number of edges (and side faces) in a faceted approximation to a true frustum
- **tol**  The frustum tolerance; error in the faceted approximation to a frustum

DESCRIPTION

This function is used to get the dimensions of the given frustum from the CIMServer workcell. The node requires the GEOMETRY attribute. A frustum may be thought of as a cone with its tip removed, or as a cylinder where one end has a smaller diameter than the other.

Polygons are used to approximate the frustum surfaces. The higher the resolution or the lower the tolerance, the closer the approximation is to a true frustum. Resolution and tolerance are dependent. Once a resolution is given for a frustum with known radii, tolerance is calculated, and vice versa. See CxGetCone for more information about the relationship between resolution, tolerance, and radius.

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>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_ATTRIBUTE_NOT_FOUND</td>
<td>The node does not have the GEOMETRY attribute.</td>
</tr>
<tr>
<td>CX_MISMATCHED_GEOM_TYPE</td>
<td>The given element is not of type FRUST (frustum).</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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>The given node does not exist.</td>
</tr>
</tbody>
</table>
EXAMPLE

The following example will get the dimensions of a frustum named \texttt{ele} and changes it to a cone with the same length as the frustum and a radius equal to the lower radius of the frustum.

```
CXServer Server;
CxNodeId ele;
double r1, r2, len, tol;
long flg, res;

/* get existing frustum dimensions from cell model */
CxGetFrust( Server, ele, &r1, &r2, &len, &flg, &res, &tol );

/* change it into a cone */
CxMakeCone( Server, ele, r1, len, flg, &res, &tol );
```

SEE ALSO

CxMakeFrust, CxGetCone
CxGetGeomType

Gets the geometry type for a node with the geometry attribute

SYNOPSIS

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

ARGUMENTS

Server  The Server ID
node    The node ID
type    The geometry type

DESCRIPTION

This function returns the geometry type for a node having the geometry attribute. Return values are for the Constructive Solid Geometry (CSG) primitives (CX_BOX, CX_CYL, CX_CONE, CX_FRUST, CX_TRAP, CX_HEMI), the swept elements (CX_SWEPTT, CX_SWEPTR), Boolean operations (CX_BOOLEAN), the general curve segment linked list of curve segments (CX_CURVE), and the polyhedron read from a neutral file such as IGES (CX_POLY). If a node has the GEOMETRY attribute but no GEOMETRY has been assigned, the type is a CX_TRIAD. The node requires 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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The node does not have the GEOMETRY attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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>

EXAMPLES

The following example changes the dimension of an element if it is of the CX_BOX type; otherwise, no action occurs.

    #include <code/robpac.h>
    .
    .
    CxServer Server;
    CxNodeId ele;
    long type;
    CxGetGeomType(Server, ele, &type);
/* if type is CX_BOX, change dimensions to 10., 20., 30. */
if (type == CX_BOX)
    CxMakeBox(Server, ele, 10., 20., 30.);

SEE ALSO

CxMakeBox
**CxGetHemi**

Gets the dimensions of a given hemisphere from the CIMServer workcell

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxGetHemi(CxServer Server, CxNodeId node, double *radius, long *res_flag, long *res, double *tol)
```

**ARGUMENTS**

- **Server**
  - The Server ID
- **node**
  - The node ID
- **radius**
  - The hemisphere radius
- **res_flag**
  - The flag which determines whether res or tol is used for hemisphere resolution. When set to **CX_ON** (1), res is used to determine tol. When set to **CX_OFF** (0), tol is used to determine res.
- **res**
  - The hemisphere resolution; the number of edges in a faceted approximation to a true hemisphere base face
- **tol**
  - The hemisphere tolerance; the error in the faceted approximation to a true hemisphere base face

**DESCRIPTION**

This function is used to get the dimensions of the given hemisphere from the CIMServer workcell. The node requires the **GEOMETRY** attribute.

Polygons are used to approximate the hemisphere surfaces. The higher the resolution or the lower the tolerance, the closer the approximation is to a true hemisphere. Resolution and tolerance are interdependent. Once a resolution is given for a hemisphere with known radius, tolerance is calculated, and vice versa. See CxGetCone for more information about the relationship between resolution, tolerance, and radius.

**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><strong>CX_MESSAGE_SEND_FAILED</strong></td>
<td>An error in sending the message.</td>
</tr>
<tr>
<td><strong>CX_MESSAGE_RECEIVE_FAILED</strong></td>
<td>An error in receiving the message.</td>
</tr>
<tr>
<td><strong>CX_MISMATCHED_GEOM_TYPE</strong></td>
<td>The given element is not of type <strong>CX_HEMI</strong> (or hemisphere).</td>
</tr>
<tr>
<td><strong>CX_ATTRIBUTE_NOT_FOUND</strong></td>
<td>The node does not have <strong>GEOMETRY</strong> attribute.</td>
</tr>
<tr>
<td><strong>CX_NODE_IS_CUT_OUT</strong></td>
<td>The given node has been cut out.</td>
</tr>
<tr>
<td><strong>CX_INVALID_NODE_ID</strong></td>
<td>The node number does not match the given node ID number.</td>
</tr>
<tr>
<td><strong>CX_NODE_NOT_FOUND</strong></td>
<td>The given node does not exist.</td>
</tr>
</tbody>
</table>
EXAMPLE

The following example will get the dimensions of a hemisphere named hemisphere. It then turns into a hemisphere with half the radius of the original.

    CxServer Server;
    CxNodeId hemi;
    double r, tol;
    long flg, res;

    /* get the dimensions of the existing hemisphere */
    CxGetHemi( Server, hemi, &r, &flg, &res, &tol );

    /* set its radius to half of its original */
    r /= 2.0;

    /* remake the hemisphere */
    CxMakeHemi( Server, hemi, r, flg, &res, &tol );

SEE ALSO

CxMakeHemi, CxGetCone
CxGetNumCurveSegs

Gets the number of parent level segments in a curve segment

SYNOPSIS

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

ARGUMENTS

- **Server**: The Server ID
- **node**: The curve node ID
- **num_segs**: The number of parent segments in a curve segment

DESCRIPTION

This function will get the number of parent level segments in a curve or path. The node requires 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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_MISMATCHED_GEOM_TYPE</td>
<td>The node is not of type CX_CURVE or CX_SWEPTR or CX_SWEPTT.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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>The node does not have GEOMETRY attribute.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following example creates a path by adding a few segments to the path. It then calls a function to query the total number of segments currently in the path and prints out the result.

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

void main(void)
{
    CxServer   Server;
    CxNodeId  path;
    long     total_segments;
```
/* Add new segments to the path */
CxAddCurveSeg( Server, path, "", "pt1", "ZXY",
               0.0, 0.0, 0.0, 0.0, 0.0, 0.0);
CxAddCurveSeg( Server, path, "pt1", "pt2", "ZXY",
               0.0, 0.0, 0.0, 250.0, 0.0, 0.0);

/* Count total number of segments comprising the path */
CxGetNumCurveSegs( Server, path, &total_segments );
printf(" Total Segments : %ld \n", total_segments );

SEE ALSO
CxGetCurveSegType, CxGetCurveSegName
**CxGetSweepEle**

Gets the sweep element type and the sweep parameters

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxGetSweepEle(CxServer Server, CxNodeId node, long *type, double *dist, long *cap_flag, long *res_flag, long *res, double *tol)
```

**ARGUMENTS**

- **Server** The Server ID
- **node** The node ID
- **type** The type `CX_SWEPTT` for extrude, or `CXSWEPTR` for revolve
- **dist** The sweep distance (length for `CX_SWEPTT` or angle for `CXSWEPTR`)
- **cap_flag** The flag which shows whether it is a solid element (`CX_TRUE`) or not (`CX_FALSE`)
- **res_flag** The flag which shows whether the resolution (`CX_ON` or 1), or the tolerance (`CX_OFF` or 0) is used
- **res** The resolution which sets the number of faceted edges
- **tol** The tolerance which sets the error in a faceted representation

**DESCRIPTION**

This function returns the type of swept element, the sweep distance, whether the element is solid (`cap_flag = CX_TRUE`), and the resolution parameters (`res_flag`, `res`, `tol`). The node requires the GEOMETRY attribute. See `CxGetCone` for more information about the relationship between resolution, tolerance, and radius.

**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><code>CX_MESSAGE_SEND_FAILED</code></td>
<td>An error in sending the message.</td>
</tr>
<tr>
<td><code>CX_MESSAGE.Receive_FAILED</code></td>
<td>An error in receiving the message.</td>
</tr>
<tr>
<td><code>CX_NODE_NOT_FOUND</code></td>
<td>The given node does not exist.</td>
</tr>
<tr>
<td><code>CX_MISMATCHED_GEOM_TYPE</code></td>
<td>The node is not of type <code>CX_SWEPTT</code> or <code>CXSWEPTR</code>.</td>
</tr>
<tr>
<td><code>CX_ATTRIBUTE_NOT_FOUND</code></td>
<td>The node does not have GEOMETRY attribute.</td>
</tr>
<tr>
<td><code>CX_NODE.IS.CUT.OUT</code></td>
<td>The 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>

**EXAMPLE**

The following example makes an inquiry about the setting of the sweep element and prints out the result.
```c
#include <stdio.h>
#include <code/robpac.h>

void main(void)
{
    CxServer   Server;
    CxNodeId  threeD_object;
    double   angle, ang_inc;
    double   sweep_dist, tolerance;
    long     object_type, is_solid;
    long     is_res, resolution;
    
    CxGetSweepEle( Server, threeD_object, &object_type, &sweep_dist, &is_solid, &is_res, &resolution, &tolerance);
    switch (object_type) {
        case CX_SWEPTT :
            printf(" Object is extruded object. 
" );
            printf(" Extrude distance is: \%f 
", sweep_dist);
            break;
        case CX_SWEPTR :
            printf(" Object is revolved object. 
" );
            printf(" Angle of revolution is: \%f 
", sweep_dist);
            break;
    }

    if (is_solid)
        printf(" Object is solid \n" );
    else
        printf(" Object is hollow \n" );

    if (is_res)
        printf(" Resolution is set to: \%ld \n", resolution );
    else
        printf(" Tolerance is set to: \%f \n", tolerance );
}

SEE ALSO
    CxMakeSweepEle, CxGetCone
```
**CxGetSweepFrame**

Gets the sweep frame position and orientation

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxGetSweepFrame(CxServer Server, CxNodeId node, char axes[4],
                      CxVector angles, CxVector vec)
```

**ARGUMENTS**

- **Server** The Server ID
- **node** The node ID
- **axes** The principal body axes about which the frame is rotated and the order of rotation (e.g. ZXY)
- **angles** The relative rotation angles in the order as the axes were returned
- **vec** The relative position vector as XYZ components

**NOTE:** Type `CxVector` is defined in `<code/matx_defs.h>` as follows:

```c
typedef double CxVector[3];
```

**DESCRIPTION**

This function gets the pose components (position and orientation) of the sweep frame relative to the node frame. The sweep frame is the reference frame for the curve to be swept. The sweep direction is along the z axis of this sweep frame for `CX_SWEPTT` types and about the z axis for `CX_SWEPTR` types (see `CxMakeSweepEle` for more information on sweep directions, etc.).

**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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_MISMATCHED_GEOM_TYPE</td>
<td>The node is not of type <code>CX_SWEPTT</code> or <code>CX_SWEPTR</code>.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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 gets the position and orientation value of the sweep node with respect to a node which specifies that a 3D path and prints out the values.

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

void main(void)
{
    CxServer  Server;
    CxNodeId  threeD_path;
    CxVector  orientation, position;

    /* Get position of sweep frame with respect to the node corresponding to the 3D path */
    CxGetSweepFrame( Server, threeD_path, "XYZ", orientation, position );
    printf(" Current sweep position is : %f %f %f \n",
            position[0], position[1], position[2] );
    printf(" Current sweep orientation is : %f %f %f \n\n",
            orientation[0], orientation[1], orientation[2] );

    ...
}

SEE ALSO
CxSetSweepFrame, CxMakeSweepEle
CxGetTrap

Gets the dimensions of a given trapezoid from the workcell

SYNOPSIS
#include <code/robpac.h>
long CxGetTrap(CxServer Server, CxNodeId node, double *L1, double *W1, double *L2, double *W2, double *h)

ARGUMENTS
Server  The Server ID
node    The node ID
L1      The trapezoid upper length
W1      The trapezoid upper width
L2      The trapezoid lower length
W2      The trapezoid lower width
h       The trapezoid width

DESCRIPTION
This function is used to get the dimensions of the given trapezoid from the CIMServer workcell. The node requires 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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_MISMATCHED_GEOM_TYPE</td>
<td>The given element is not of type TRAP (or trapezoid).</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The node does not have the GEOMETRY attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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 is a complete CODE application process. When compiled and linked properly, it will run with the CIMServer and accomplish the tasks described below.

The process makes a box ele1 sitting on top of an existing trapezoid ele. The box will have the same length and width as the upper length and width of the trapezoid.

The CxGetTrap function will be used to determine the dimensions of the existing trapezoid, while CxAddNewNode, CxSetRelPose, CxAddGeometry, and CxMakeBox are called (in that order) to make the box.
#include <stdio.h>
#include <code/robpac.h>

void main(void)
{
    CxServer Server;
    double l1, w1, l2, w2, h;
    CxVector ang, vec;
    long i;
    CxNodeId ele, ele1;

    /* initialization */
    Server = CxOpenServer("Sample", CX_SMEM, 0);
    CxErrorPolicy(CX_RETURN_ERRORS);
    CxSetErrorLogPolicy (CX_LOG_TO_TTY, CX_NULL, 0);

    /* get existing trapezoid dimensions from cell model */
    CxGetNamedNodeId ( Server, "ele", &ele );
    CxGetTrap( Server, ele, &l1, &w1, &l2, &w2, &h);

    /* add new node named ele1 as a child of ele */
    CxAddNewNode( Server, "ele1", ele );
    CxGetNamedNodeId ( Server, "ele1", &ele1 );

    /* set relative pose for new node "ele" */
    for(i=0; i<3; i++) ang[i] = 0.0;
    vec[0] = (l2 - l1) / 2.0;
    vec[1] = (w2 - w1) / 2.0;
    vec[2] = h;
    CxSetRelPose(Server, ele1, "ZYX", ang, vec);

    /* add geometry attribute to "ele1" */
    CxAddGeometry( Server, ele1);

    /* make the box sitting on top of the trapezoid */
    if( CxMakeBox( Server, ele1, l1, w1, 100. ) == CX_ERROR) {
        fprintf(stderr,"CX_ERROR making box...
");
    } else {
        /* change box’s color to RED */
        CxChangeColor( Server, ele1, 1.0, 0.0, 0.0);
    }

    /* exit */
    CxRobpacExit();
}

SEE ALSO

    CxMakeTrap
**CxMakeBoolean**

Makes a new solid by the Boolean operation of two solids

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxMakeBoolean( CxServer Server, CxNodeId node, CxNodeId node_a, 
                   long operation, CxNodeId node_b )
```

**ARGUMENTS**

- **Server** The Server ID
- **node** The element of the Boolean result
- **node_a** The element of the Boolean component a
- **Operation** The Boolean operations to be used: CX_SUB, CX_ADD, or CX_INTRX
  - **CX_SUB**: The difference between two solids
  - **CX_ADD**: The union of two solids
  - **CX_INTRX**: The intersection of the two Boolean components
- **node_b** An element of the Boolean component b

**DESCRIPTION**

This function is used to make a new solid (Boolean result) by Boolean operation of two solids. The result node and the component nodes require 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>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_ATTRIBUTE_NOT_FOUND</td>
<td>The Boolean result or Boolean component nodes do not have the GEOMETRY attribute.</td>
</tr>
<tr>
<td>CX_BOOLEAN_FAILED</td>
<td>The Boolean operation failed.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>Any 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>Any given node does not exist.</td>
</tr>
<tr>
<td>CX_MACHINE_OUT_OF_MEMORY</td>
<td>A memory allocation error.</td>
</tr>
<tr>
<td>CX_ELEMENT_NOT_FOUND</td>
<td>The given node element does not exist.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following example is a complete CODE application process. When compiled and linked properly, the code will run with the CIMServer and accomplish the desired task described below.
This process first makes a box as a child of the world node. Another node named cyl is then added as a child of the world node, also. The cyl node is posed at the center of the box, using the function CxSetRelPose. The Boolean result node bool is then added with the GEOMETRY attribute, and CxMakeBoolean is used to make the geometry as the result of subtracting cyl from box, or bool = box - cyl.

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

void main(void)
{
    CxServer Server;
    CxVector ang, vec;
    long i;
    double tol = 0.2;
    long res = 12;
    CxNodeId world, box, cyl, bool;

    /* initialize and set run status */
    Server = CxOpenServer("Sample", CX_SMEM, 0);
    CxErrorPolicy(CX_RETURN_ERRORS);
    CxSetErrorLogPolicy (CX_LOG_TO_TTY, CX_NULL, 0);

    /* add new node named box as a child of world */
    CxGetNamedNodeId ( Server, "world", &world );
    CxAddNewNode( Server,"box", world );
    CxGetNamedNodeId ( Server, "box", &box );
    CxAddGeometry( Server, box );
    CxMakeBox( Server, box, 100., 100., 100. );

    /* add new node named cyl as a child of world */
    CxAddNewNode( Server, "cyl", world );
    CxGetNamedNodeId ( Server, "cyl", &cyl );
    CxAddGeometry( Server, cyl );

    /* position the cyl at the center of the box */
    for(i=0; i<3; i++) {
        ang[i] = 0.0;
    }
    vec[0] = 50.;
    vec[1] = 50.;
    vec[2] = -50.;
    CxSetRelPose( Server, cyl, "XYZ", ang, vec);
    CxMakeCyl( Server, cyl, 30., 200., 1, &res, &tol);

    /* add new node named bool as a child of world */
    CxAddNewNode( Server, "bool", world );
    CxGetNamedNodeId ( Server, "bool", &bool );
    CxAddGeometry( Server, bool );

    /* make bool as the result of box subtract cyl */
    CxMakeBoolean( Server, bool, box, CX_SUB, cyl);
}```
SEE ALSO

CxGetBoolean
CxMakeBox

Makes a box of the given dimensions

SYNOPSIS

```
#include <code/robpac.h>
long CxMakeBox(CxServer Server, CxNodeId node, double L1, double L2,
              double L3)
```

ARGUMENTS

- **Server**: The Server ID
- **Node**: The node ID
- **L1**: The box dimension in local x direction
- **L2**: The box dimension in local y direction
- **L3**: The box dimension in local z direction

DESCRIPTION

This function is used to make a box with the given dimensions. The node requires the \texttt{GEOMETRY} attribute.

RETURN VALUES

This function returns \texttt{0} if successful; otherwise, \texttt{-1 (CX\_ERROR)} is returned.

ERRORS

If the function returns an error condition, the error code can be obtained by using the \texttt{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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the \texttt{GEOMETRY} attribute.</td>
</tr>
<tr>
<td>CX_MACHINE_OUT_OF_MEMORY</td>
<td>A memory allocation error.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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 is a complete CODE application process. When compiled and linked properly, it will run with the CIMServer and accomplish the tasks described below.

The process creates a box 400 by 300 by 100 units, with the box centered over the world node frame. The \texttt{CxAddNewNode} function (See the \texttt{Nodes, Frames and Attributes} section of the \texttt{CODE API Programmer’s Reference Manual - Volume 2}) creates a new node named \texttt{base\_box} that is a child of the \texttt{world} node, while \texttt{CxSetRelPose} (See the \texttt{Geometry} section of the \texttt{CODE API Programmer’s Reference Manual – Volume 2}) sets the position and orientation of \texttt{base\_box} relative to its parent. The function \texttt{CxAddGeometry} (See the \texttt{Nodes, Frames and Attributes} section of the \texttt{CODE API Programmer’s Reference Manual - Volume 2}) gives the current node the geometry attribute.
```c
#include <stdio.h>
#include <code/robpac.h>

void main(void)
{

    CxServer Server
    double L1, L2;
    CxVector ang, vec;
    long i;
    CxNodeId world, box;

    /* initialize and set run status */
    Server = CxOpenServer("Sample", CX_SMEM, 0);
    CxErrorPolicy(CX_RETURN_ERRORS);
    CxGetNamedNodeId ( Server, "world", &world );

    /* add a node named base_box as a child of world */
    CxAddNewNode( Server, "base_box", world );
    CxGetNamedNodeId ( Server, "base_box", &box );

    /*set relative pose for the base_box */
    L1 = 400.0;
    L2 = 300.0;

    for(i=0; i<3; i++) ang[i] = 0.0;
    vec[0] = - L1 / 2.0;
    vec[1] = - L2 / 2.0;
    vec[2] = 0.0;
    CxSetRelPose( Server, box, "ZYX", ang, vec);

    /* add geometry attribute to node base_box */
    CxAddGeometry( Server, box );

    /* make a box of the given dimensions */
    if(CxMakeBox( Server, box, L1, L2, 100.) == CX_ERROR) {
        fprintf(stderr,"CX_ERROR making base_box...
"");
    }

    /* exit */
    CxRobpacExit();
}

SEE ALSO
    CxGetBox
```
CxMakeCone

Makes a cone of the given dimensions

SYNOPSIS

#include <code/robpac.h>
long CxMakeCone(CxServer Server, CxNodeId node, double radius, double
length, long res_flag, long *res, double *tol)

ARGUMENTS

Server  The Server ID
Node    The node ID
Radius  The cone radius
Length  The cone length
res_flag The flag which determines whether res or tol is used for cone resolution. When set to
CX_ON (1), res is used to determine tol. When set to CX_OFF (0), tol is used to determine res.
res     The cone resolution; the number of edges (and side faces) in a faceted approximation to a
true cone
tol     The cone tolerance; the error in the faceted approximation to a true cone

DESCRIPTION

This function is used to make a cone of given dimensions. The node requires the GEOMETRY attribute.
Polygons are used to approximate the cone surfaces. The higher the resolution or the lower the tolerance, the
closer the approximation is to a true cone. Resolution and tolerance are dependent. Once a resolution is given
for a cone with known radius, tolerance is calculated, and vice versa. See CxGetCone for more information
about the relationship between resolution, tolerance, and radius.

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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the GEOMETRY attribute.</td>
</tr>
<tr>
<td>CX_MACHINE_OUT_OF_MEMORY</td>
<td>A memory allocation error.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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 is a complete CODE application process. When compiled and linked properly, it will
run with the CIMServer, and accomplish the tasks described below.
The process will make a cone of radius 100 units and length 250 units. A 16-edged polygon will be used to approximate the circular aspect of the cone (resolution equals 16). The node `ele` is added as a child of the `world` node. The program then gives the geometry attribute to `ele` and makes the desired cone.

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

void main(void)
{
    CxServer Server;
    double tol, rad, len;
    long res = 16;
    CxNodeId world, ele;

    /* initialize and set run status */
    Server = CxOpenServer("Sample", CX_SMEM, 0);
    CxErrorPolicy(CX_RETURN_ERRORS);
    CxSetErrorLogPolicy (CX_LOG_TO_TTY, CX_NULL, 0);

    CxGetNamedNodeId ( Server, "world", &world );

    /* add new node named ele as a child of world */
    CxAddNewNode( Server, "ele", world );
    CxGetNamedNodeId ( Server, "ele", &ele );

    /* add geometry attribute to "ele" */
    CxAddGeometry( Server, ele );

    /* make the cone */
    if(CxMakeCone( Server, ele, 100., 250., CX_ON, &res, &tol ) ==
       CX_ERROR) {
        fprintf(stderr,"CX_ERROR making cone...\n");
    }

    /* exit */
    CxRobpacExit();
}

SEE ALSO

CxGetCone
CxMakeCyl

Makes a cylinder of the given dimensions

SYNOPSIS

```
#include <code/robpac.h>
long CxMakeCyl(CxServer Server, CxNodeId node, double radius, double length, long res_flag, long *res, double *tol)
```

ARGUMENTS

- **Server**  
The Server ID
- **node**  
The node ID
- **radius**  
The cylinder radius
- **length**  
The cylinder length
- **res_flag**  
The flag which determines whether res or tol is used for cylinder resolution. When set to CX_ON (1), res is used to determine tol. When set to CX_OFF (0), tol is used to determine res.
- **res**  
The cylinder resolution; the number of edges (and side faces) in a faceted approximation to a true cylinder
- **tol**  
The cylinder tolerance; the error in the faceted approximation to a true cylinder

DESCRIPTION

This function is used to make a cylinder of the given dimensions. The node requires the GEOMETRY attribute. Polygons are used to approximate the cylinder surfaces. The higher the resolution or the lower the tolerance, the closer the approximation is to a true cylinder. Resolution and tolerance are interdependent. Once a resolution is given for a cylinder with known radius, tolerance is calculated, and vice versa. See CxGetCone for more information about the relationship between resolution, tolerance, and radius.

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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the GEOMETRY attribute.</td>
</tr>
<tr>
<td>CX_MACHINE_OUT_OF_MEMORY</td>
<td>A memory allocation error.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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 will make a cylinder of radius 50 units and length 100 units. The facet approximation tolerance is 0.2. The node cylinder must already exist in the workcell cell model and must already have the geometry attribute.

```c

Cylinder Server;
Cylinder cyl;
double tol = 0.2;
long res;

CxMakeCyl( Server, cyl, 50., 100., CX_OFF, &res, &tol);

SEE ALSO

CxGetCyl, CxGetCone
```
CxMakeFrust

Makes a frustum of the given dimensions

SYNOPSIS
#include <code/robpac.h>
long CxMakeFrust(CxServer Server, CxNodeId node, double R1, double R2,
    double L, long res_flag, long *res, double *tol)

ARGUMENTS
Server             The Server ID
node               The node ID
R1                 The frustum upper radius
R2                 The frustum lower radius
L                  The frustum length
res_flag           The flag which determines whether res or tol is used for frustum resolution. When set to
                   CX_ON (1), res is used to determine tol. When set to CX_OFF (0), tol is used to determine
                   res.
res                The frustum resolution; the number of edges (and side faces) in a faceted approximation
                   to a true frustum
tol                The frustum tolerance; the error in the faceted approximation to a true frustum

DESCRIPTION
This function is used to make a frustum of given dimensions. The node requires the GEOMETRY attribute. A
frustum may be thought of as a cone with its tip removed, or as a cylinder where one end has a smaller
diameter than the other.
Polygons are used to approximate the frustum surfaces. The higher the resolution or the lower the tolerance,
the closer the approximation is to a true frustum. Resolution and tolerance are interdependent. Once a
resolution is given for a frustum with known radius, tolerance is calculated, and vice versa. See CxGetCone
for more information about the relationship between resolution, tolerance, and radius.

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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the GEOMETRY attribute.</td>
</tr>
<tr>
<td>CX_MACHINE_OUT_OF_MEMORY</td>
<td>A memory allocation error.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The given node has been cut out.</td>
</tr>
<tr>
<td>Error Codes</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------------------</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 gets the parameters of the frustum f1 from the workcell file using CxGetFrust. The tolerance is then set to be half of the original tolerance, and the frustum is recreated with the new tolerance using CxMakeFrust.

```c

CxServer Server;
CxNodeId f1;
double r1, r2, h, tol;
long res, flg;

/* get existing frustum dimensions from the cell model */
CxGetFrust( Server, f1, &r1, &r2, &h, &flg, &res, &tol );

/* set tighter facet approximation tolerance */
tol /= 2.0;
CxMakeFrust( Server, f1, r1, r2, h, CX_OFF, &res, &tol );
```

**SEE ALSO**

CxGetFrust, CxGetCone
CxMakeHemi

Makes a hemisphere of given dimensions

SYNOPSIS

#include <code/robpac.h>
long CxMakeHemi( CxServer Server, CxNodeId node, double radius, long
res_flag, long *res, double *tol )

ARGUMENTS

Server  The Server ID
node     The node ID
radius   The hemisphere radius
res_flag The flag which determines whether res or tol is used for hemisphere resolution. When set to
CX_ON (1), res is used to determine tol. When set to CX_OFF (0), tol is used to determine
res.
res      The hemisphere resolution; the number of edges in a faceted approximation due to a true
hemisphere base face
tol      The hemisphere tolerance; the error in the faceted approximation to a true hemisphere base
face

DESCRIPTION

This function is used to make a hemisphere of given dimensions. The node requires the GEOMETRY attribute.
Polygons are used to approximate the hemisphere surfaces. The higher the resolution or the lower the
tolerance, the closer the approximation is to a true hemisphere. Resolution and tolerance are interdependent.
Once a resolution is given for a hemisphere with known radius, tolerance is calculated, and vice versa. See
CxGetCone for more information about the relationship between resolution, tolerance, and radius.

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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have the GEOMETRY attribute.</td>
</tr>
<tr>
<td>CX_MACHINE_OUT_OF_MEMORY</td>
<td>A memory allocation error.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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 uses CxGetCyl to get the dimensions of a cylinder named ele. The geometry of ele is then redefined as a hemisphere of the same radius and resolution as the original cylinder using CxMakeHemi.

```c
CxServer Server;
CxNodeId ele;
double r, l, tol;
long flg, res;

/* get the dimensions of "ele", which is a cylinder */
CxGetCyl( Server, ele, &r, &l, &flg, &res, &tol );

/* make "ele" a hemisphere, with the same radius and resolution as the cylinder */
CxMakeHemi( Server, ele, r, flg, &res, &tol );
```

SEE ALSO

CxGetHemi, CxGetCone
**CxMakeSweepEle**

Sets the sweep type and the sweep parameters

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxMakeSweepEle(CxServer Server, CxNodeId node, long type, double dist, long cap_flag, long res_flag, long *res, double *tol)
```

**ARGUMENTS**

- **Server** The Server ID
- **node** The node ID
- **type** The type (CX_SWEPTT for extrude, or CXSWEPTR for revolve)
- **dist** The sweep distance (angle for CXSWEPTR, or length for CX_SWEPTT)
- **cap_flag** The flag which determines whether the cap_flag element is solid (CX_TRUE), or not (CX_FALSE)
- **res_flag** The flag which determines whether resolution (CX_ON or 1), or tolerance (CX_OFF or 0) is used
- **res** The resolution which sets the number of faceted edges
- **tol** The tolerance which sets the error in a faceted representation

**DESCRIPTION**

This function sets the type of swept element and the sweep distance. In addition, the user can set the cap_flag to CX_TRUE if the swept element is to be capped and made solid (if the constituent curve segment is closed). If the type is CXSWEPTR, then the user must enter parameters which define the faceted resolution of the revolved element. Since these parameters may be modified internally (due to resolution limitations), their address must be passed through. The node requires the GEOMETRY attribute.

Polygons are used to approximate the hemisphere surfaces. The higher the resolution or the lower the tolerance, the closer the approximation is to a true hemisphere. Resolution and tolerance are interdependent. Once a resolution is given for a hemisphere with known radius, tolerance is calculated, and vice versa. See CxGetCone for more information about the relationship between resolution, tolerance, and radius.

This function will not work unless the element has a CX_CURVE attribute previously defined for it.

**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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The node does not have the GEOMETRY attribute.</td>
</tr>
<tr>
<td>Error Codes</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CX_MISMATCHED_GEOM_TYPE</td>
<td>The node is not of type CX_CURVE or CX_SWEPT or CX_SWEPTR.</td>
</tr>
<tr>
<td>CX_NO_EDGE_TO_SWEEP</td>
<td>No curve segments exist.</td>
</tr>
<tr>
<td>CX_EDGE_ON_REVOLVE_AXIS</td>
<td>The edge cannot be revolved about itself.</td>
</tr>
<tr>
<td>CX_SWEEP_EDGES_INTX</td>
<td>The edges cross each other.</td>
</tr>
<tr>
<td>CX_NON_COPLANAR_VERTICES</td>
<td>The curve segments do not lie in plane.</td>
</tr>
<tr>
<td>CX_SWEEP_VECTOR_PARAL_TO_EDGE</td>
<td>The extrude direction must be normal to plane of curve edges.</td>
</tr>
<tr>
<td>CX_SWEEP_DISTANCE_ZERO</td>
<td>The sweep distance Must be non-zero.</td>
</tr>
<tr>
<td>CX_REVOLVE_AXIS_NORMAL_TO_EDGES</td>
<td>The sweep z axis must be parallel to the plane of curve edges.</td>
</tr>
<tr>
<td>CX_SWEEP_EDGE_INTX_SWEEP_AXIS</td>
<td>The curve edges cannot cross the sweep z axis for CX_SWEPT.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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_INVALID_ARGUMENT</td>
<td>The cap_flag must be either 1 or 0.</td>
</tr>
<tr>
<td>CX_REVOLVE_AXIS_ NOT_PARAL_TO_EDGES</td>
<td>The revolving axis is not parallel to the edges.</td>
</tr>
</tbody>
</table>

**EXAMPLES**

The following example makes a solid swept element of the type CX_SWEPTT by extruding a curve segment through a distance of 100, with cap on. A resolution of 10 is used.

```plaintext

. .
CxServer Server;
CxNodeId ele;
long res_flag = CX_TRUE;
long res = 10;
double tol;
.
.
CxMakeSweepEle(Server,ele,CX_SWEPTT,100.,CX_TRUE, res_flag, &res, &tol);
.
.
SEE ALSO

CxGetSweepEle, CxGetCone
```
**CxMakeTrap**

Makes a trapezoid of the given dimensions

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxMakeTrap(CxServer Server, CxNodeId node, double L1, double W1, double L2, double W2, double h)
```

**ARGUMENTS**

- **Server**: The Server ID
- **node**: The node ID
- **L1**: The trapezoid upper length
- **W1**: The trapezoid upper width
- **L2**: The trapezoid lower length
- **W2**: The trapezoid lower width
- **h**: The trapezoid height

**DESCRIPTION**

This function is used to make a trapezoid of the given dimensions. The node requires 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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The given node does not have <strong>GEOMETRY</strong> attribute.</td>
</tr>
<tr>
<td>CX_MACHINE_OUT_OF_MEMORY</td>
<td>A memory allocation error.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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 is a complete CODE application process.

The code makes a trapezoid which sits on top of an existing box. The trapezoid trap will have an upper length and width double the corresponding dimensions on the box, and a lower length and width the same as the box. The height of the trapezoid will be twice that of the box. The **CxGetBox** function will be used to determine the dimensions of the existing box, while **CxAddNewNode**, **CxAddGeometry**, and
CxMakeTrap are called (in that order) to make the trapezoid. If the trapezoid is successfully made, its color will be set to BLUE with CxChangeColor (from the “Render” library).

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

void main(void)
{
    CxServer Server;
    double x, y, z, l1, w1, l2, w2, h;
    CxVector ang, vec;
    long i;
    CxNodeId box, trap;

    /* initialize */
    Server = CxOpenServer("Sample", CX_SMEM, 0);
    CxSetErrorPolicy(CX_RETURN_ERRORS);
    CxGetNamedNodeId ( Server, "box", &box );

    /* get existing box dimensions from cell model */
    if(CxGetBox ( Server, box, &x, &y, &z ) == CX_ERROR) {
        fprintf(stderr, "box does not exist...
"       
            CxRobpacExit();
    }

    /* add new node named trap as a child of box */
    CxAddNewNode ( Server, "trap", box );
    CxGetNamedNodeId ( Server, "trap", &trap );

    /* set relative pose for new node "trap" */
    for(i=0; i<3; i++) {
        ang[i] = 0.0;
        vec[i] = 0.0;
        vec[2] = z;
    }
    CxSetRelPose ( Server, trap, "ZYX", ang, vec );

    /* add geometry attribute to "trap" */
    CxAddGeometry ( Server, trap );

    /* calculate trapezoid dimensions */
    l1 = x * 2.0;
    w1 = y * 2.0;
    l2 = x;
    w2 = y;
    h = 2. * z;
    /* make the trapezoid sitting on top of the box */
    if(CxMakeTrap ( Server, trap, l1, w1, l2, w2, h ) == CX_ERROR) {
        fprintf(stderr, "CX_ERROR making trapezoid...
"
    } else {
        /* change trapezoid’s color to BLUE */
```
CxChangeColor( Server, trap, 0., 0., 1. );
}

/* exit */
CxRobpacExit();

SEE ALSO
   CxGetTrap
CxSetCurveSegArc

Sets a curve segment type to circular

SYNOPSIS

#include <code/robpac.h>
long CxSetCurveSegArc( CxServer Server, CxNodeId node, char *seg_name, 
long axis_num)

ARGUMENTS

Server       The Server ID
node         The curve node ID
seg_name     The name of the segment to be set to type CX_CIRCULAR_INTERP
axis_num     The number of the axis tangent to the circular arc (0, 1, or 2 where 0 = x, 1 = y, 2 = z)

DESCRIPTION

This function allows the user to specify a named curve segment as a CX_CIRCULAR_INTERP type, unless it is the terminal frame. If the terminal frame is used, this function will be ignored. If seg_name is entered as " " (the empty string), the first segment will be reset to circular type. The node requires 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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_NO_SUCH_CURVE_SEG_NAME</td>
<td>The given node is not of type CX_CURVE or no segments of the given name exists.</td>
</tr>
<tr>
<td>CX_NON_XYZ_AXIS</td>
<td>The axis_num is not 0, 1, or 2</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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_MISMATCHED_GEOM_TYPE</td>
<td>The node is not of type CX_CURVE or CX_SWEPT or CX_SWEPTR</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The node does not have the GEOMETRY attribute.</td>
</tr>
</tbody>
</table>

EXAMPLE

The example makes a path by adding 6 curve segments (named pt1 through pt6). Curve segment pt4 of the path is then given a CIRCULAR type.
#include <stdio.h>
#include <code/robpac.h>

void main(void)
{
    CxServer Server,
    CxNodeId path;
    double  tol, axis[4];
    char    seg[CX_MAXNAME];

    Server = CxOpenServer("Path", CX_SMEM, 0);
    CxGetNamedNodeId( Server, "path", &path );

    /* add the six curve segments */
    CxAddCurveSeg( Server, path, "pt1", "pt1", "ZYX", 270., 0., 0.,
                   0., 0., 0.);
    CxAddCurveSeg( Server, path, "pt1", "pt2", "ZYX", 0., 0., 0.,
                   250., 0., 0.);
    CxAddCurveSeg( Server, path, "pt2", "pt3", "ZYX", 90., 0., 0.,
                   250., 450., 0.);
    CxAddCurveSeg( Server, path, "pt3", "pt4", "ZYX", 270., 0., 0.,
                   0., 375., 0.);
    CxAddCurveSeg( Server, path, "pt4", "pt5", "ZYX", 0., 0., 0.,
                   0., 100., 0.);
    CxAddCurveSeg( Server, path, "pt5", "pt6", "ZYX", 0., 0., 0.,
                   0., 0., 0.);

    CxSetCurveSegArc( Server, path, "pt4", 1 );
}

SEE ALSO
CxAddCurveSeg, CxSetCurveSegLinear, CxSetCurveSegLincirc, CxSetCurveSegArc3
CxSetCurveSegArc3

Sets a circular arc by three points

SYNOPSIS

```c
#include <code/robpac.h>
long CxSetCurveSegArc3(CxServer Server, CxNodeId node, char *seg_name, double x, double y, double z)
```

ARGUMENTS

- **Server**: The Server ID
- **node**: The curve node ID
- **seg_name**: The name of the segment to be set to type `CX_CIRCULAR_INTERP`
- **x**, **y**, **z**: The location of the arc mid-point specified relative to curve_node

DESCRIPTION

This function allows the user to specify a named curve segment as a `CX_CIRCULAR_INTERP` type, using the frame of the segment identified by `seg_name`, and a mid-arc point described by three coordinates (x, y, z). If `seg_name` is entered as "" (the empty string), the first segment will be reset to circular type. The node requires 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><code>CX_MESSAGE_SEND_FAILED</code></td>
<td>An error in sending the message.</td>
</tr>
<tr>
<td><code>CX_MESSAGE_RECEIVE_FAILED</code></td>
<td>An error in receiving the message.</td>
</tr>
<tr>
<td><code>CX_NODE_NOT_FOUND</code></td>
<td>The given node does not exist.</td>
</tr>
<tr>
<td><code>CX_NO_SUCH_CURVE_SEG_NAME</code></td>
<td>The given node is not of type <code>CX_CURVE</code> or no seg of given name exists.</td>
</tr>
<tr>
<td><code>CX_NODE_IS_CUT_OUT</code></td>
<td>The 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>
<tr>
<td><code>CX_MISMATCHED_GEOM_TYPE</code></td>
<td>The node is not of type <code>CX_CURVE</code> or <code>CX_SWEPTT</code> or <code>CX_SWEPTR</code>.</td>
</tr>
<tr>
<td><code>CX_ATTRIBUTE_NOT_FOUND</code></td>
<td>The node does not have the <code>GEOMETRY</code> attribute.</td>
</tr>
</tbody>
</table>

WARNING

Users can modify a curvilinear segment using this function, but the action is non-recoverable, unless the user uses other API functions to save and restore the segment type.

EXAMPLE

The following example adds a curve segment and then sets the segment type to circular:
CxServer Server;
CxNodeId path;

CxAddCurveSeg( Server, path, "", "pt1", "ZYX", 270., 0., 0., 0., 0.);
CxAddCurveSeg( Server, path, "pt1", "pt2", "ZYX", 0., 0., 0., 250., 0., 0.);
/* Give the segment a curve type of circular */
CxSetCurveSegArc3( Server, path, "pt1", 100., -75., 0. );

SEE ALSO

CxAddCurveSeg, CxSetCurveSegLinear, CxSetCurveSegLincirc, CxSetCurveSegArc
CxSetCurveSegFrame

Sets the frame matrix for a segment in a curve

SYNOPSIS

```c
#include <code/robpac.h>
long CxSetCurveSegFrame(CxServer Server, CxNodeId node, char *seg_name,
                        char axes[4], double ax, double ay, double az, double x, double
                        y, double z)
```

ARGUMENTS

- **Server**: The Server ID
- **node**: The node ID
- **seg_name**: The name of the segment
- **axes**: The principal axes about which the frame is rotated and the order of rotation (e.g., ZYX)
- **ax, ay, az**: The relative rotation angles in order of axes (e.g., ZYX)
- **x, y, z**: The relative position components

DESCRIPTION

This function sets the principal axes about which the frame is rotated and the order of rotation, the rotation angles, and position components which pose a segment of name `seg_name` relative to the curve node. If the name is entered as " " (the empty string), then the frame of the first segment will be set; otherwise, the frame of the segment with name `seg_name` will be set. In other words, after adding curve segments, this function lets the user edit position and orientation of the segments. The node requires 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>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 given curve node does not exist.</td>
</tr>
<tr>
<td>CX_CURVE_SEG_NOT_NAMED</td>
<td>The <code>seg_name</code> is not valid.</td>
</tr>
<tr>
<td>CX_NO_SUCH_CURVE_SEG_NAME</td>
<td>The <code>seg_name</code> 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_INVALID_NODE_ID</td>
<td>The node number does not match the given node ID number.</td>
</tr>
<tr>
<td>CX_MISMATCHED_geom_type</td>
<td>The node is not of type CX_CURVE, CX_SWEPTT, or CX_SWEPTR.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The node does not have the <code>GEOMETRY</code> attribute.</td>
</tr>
</tbody>
</table>
EXAMPLE

The following example adds a segment to the path. It then sets the type to be CX_CIRCULAR_INTERP, and rotates the reference frame of that segment through 360 degrees, effectively rotating the segment.

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

void main(void)
{
    CxServer   Server;
    CxNodeId  path;
    double   angle;
    .
    /* Add a new circular segment to the path */
    CxAddCurveSeg( Server, path, "pt2", "pt3", "ZXY",
                   0.0, 0.0, 0.0, 250.0, 0.0, 0.0);
    CxSetCurveSegArc( Server, path, "pt3", 0 );
    /* Rotate segment frame about Y axis by 360 degrees and move its position */
    for (angle=0.0; angle<361.0; angle+=10.0)
        CxSetCurveSegFrame ( Server, path, "pt3", "YZX", angle, 0.0, 0.0, 0.0, 810.0, 70.0 );
    .
}
```

SEE ALSO

CxGetCurveSegFrame, CxSetCurveSegArc
CxSetCurveSegLincirc

Specifies a curve segment type as a combination linear/circular

SYNOPSIS

#include <code/robpac.h>
long CxSetCurveSegLincirc(CxServer Server, CxNodeId node, char *seg_name, long axis_num)

ARGUMENTS

Server        The Server ID
Node          The curve node ID
Seg_name      The name of the segment to be set to type CX_LINCIRC_INTERP
Axis_num      The number of the axis tangent to the curve segment (0,1, or 2 where 0 = x, 1 = y, 2 = z)

DESCRIPTION

This function allows the user to specify a named curve segment as a CX_LINCIRC_INTERP type, unless it is the terminal frame. If the terminal frame is used, this function will be ignored. If seg_name is entered as " (the empty string), the first segment will be set to lincirc type. The node requires 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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_NO_SUCH_CURVE_SEG_NAME</td>
<td>The given node is not of type CX_CURVE, or no segment of the given name exists.</td>
</tr>
<tr>
<td>CX_NON_XYZ_AXIS</td>
<td>The axis_num is not 0, 1, or 2</td>
</tr>
<tr>
<td>CX_CURVE_DECOMP_ERROR</td>
<td>There are improper parameters for the lincirc space plane (no arc defined or arc out of plane).</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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_MISMATCHED_GEOM_TYPE</td>
<td>The node is not of type CX_CURVE, CX_SWEPTT, or CX_SWEPTR.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The node does not have the GEOMETRY attribute.</td>
</tr>
</tbody>
</table>

EXAMPLE

The following example adds a curve segment and sets the segment type to lincirc.
CxServer Server;
CxNodeId path;

CxAddCurveSeg( Server, path, "", "pt1", "ZYX", 270.0, 0.0, 0.0,
0.0,0.0, 0.0 );
CxAddCurveSeg( Server, path, "pt1", "pt2", "ZYX", 0.0, 0.0, 0.0, 0.0,
100.0, 0.0 );
/* Give the segment a curve type of CX_LINCIRC_INTERP */
CxSetCurveSegLincirc( Server, path, "pt1", 1 );

SEE ALSO
CxAddCurveSeg, CxSetCurveSegLinear, CxSetCurveSegArc, CxSetCurveSegArc3
CxSetCurveSegLinear
Sets a curve segment type as linear

SYNOPSIS
#include <code/robpac.h>
long CxSetCurveSegLinear(CxServer Server, CxNodeId node, char *seg_name)

ARGUMENTS
Server The Server ID
node The curve node ID
seg_name The name of the segment to be set to type CX_LINEAR_INTERP

DESCRIPTION
This function allows the user to specify a named curve segment as a CX_LINEAR_INTERP type, unless it is
the terminal frame. If the terminal frame is used, this function will be ignored. If seg_name is entered as " "
(the empty string), the first segment will be set to linear. The node requires 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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_NO_SUCH_CURVE_SEG_NAME</td>
<td>The given node is not of type CX_CURVE, or no segment of the</td>
</tr>
<tr>
<td></td>
<td>given name exists.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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_MISMATCHED_GEOM_TYPE</td>
<td>The node is not of type CX_CURVE, CX_SWEPTT, or CX_SWEPTR.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The node does not have the GEOMETRY attribute.</td>
</tr>
</tbody>
</table>

EXAMPLE
The following example checks the current segment type of a node. If the type is not CX_LINEAR_INTERP
it sets it to CX_LINEAR_INTERP type.

#include <code/robpac.h>

void main(void)


{   CxServer   Server;
CxNodeId path;
    long     seg_type, path_axis;
    
    /* Set segment type to be linear if not so */
    CxGetCurveSegType( Server, path, "pt3", &seg_type, &path_axis );
    if (seg_type != CX_LINEAR_INTERP)
        CxSetCurveSegLinear( Server, path, "pt3" );
    ...
}

SEE ALSO

CxAddCurveSeg, CxSetCurveSegLincirc, CxSetCurveSegArc, CxSetCurveSegArc3
**CxSetCurveSegName**

Sets (rename) a frame in a linked list of curve segments

**SYNOPSIS**

```c
#include <code/robpac.h>
long CxSetCurveSegName(CxServer Server, CxNodeId node, char *old_seg_name, char *new_seg_name)
```

**ARGUMENTS**

- **Server**: The Server ID
- **node**: The curve node ID
- **old_seg_name**: The old name of the segment
- **new_seg_name**: The new name of the segment

**DESCRIPTION**

This function renames the segment with name `old_seg_name`. If the old segment name is entered as " " (the empty string), then the name of the first segment will be renamed. The node requires 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>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 given curve node does not exist.</td>
</tr>
<tr>
<td>CX_NO_SUCH_CURVE_SEG_NAME</td>
<td>The <code>old_seg_name</code> does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The node does not have the <code>GEOMETRY</code> attribute.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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_CURVE_SEG_NOT_NAMED</td>
<td>The <code>new_seg_name</code> is not valid.</td>
</tr>
<tr>
<td>CX_REDUNDANT_CURVE_SEG_NAME</td>
<td>A curve segment of that name already exists.</td>
</tr>
<tr>
<td>CX_MISMATCHED_GEOM_TYPE</td>
<td>The node is not of type <code>CX_CURVE</code>, <code>CX_SWEPTT</code>, or <code>CX_SWEPTR</code>.</td>
</tr>
</tbody>
</table>

**EXAMPLES**

The following example checks the current segment type of a node. If the type is `CX_LINEAR_INTERP` it appends "LIN" to the name; otherwise, it appends "NOLIN" to its name.
#include <code/robpac.h>

void main(void)
{
    CxServer Server;
    CxNodeId path;
    long seg_type, path_axis;
    .
    /* Check segment type and set an appropriate name */
    CxGetCurveSegType( Server, path, "pt3", &seg_type, &path_axis );
    if (seg_type == CX_LINEAR_INTERP)
        CxSetCurveSegName( Server, path, "pt3", "pt3_LIN" );
    else
        CxSetCurveSegName( Server, path, "pt3", "pt3_NOLIN" );
    .
}

SEE ALSO

CxGetCurveSegName
CxSetSweepFrame
Sets the sweep frame position and orientation

SYNOPSIS
#include <code/robpac.h>
long CxSetSweepFrame(CxServer Server, CxNodeId node, char axes[4],
double a1, double a2, double a3, double x, double y, double z)

ARGUMENTS
Server       The Server ID
node         The node ID
axes         The principal body axes about which the frame is rotated and the order of rotation (e.g. XYZ)
a1, a2, a3   The rotation angles in order of axes
x, y, z      The position components

DESCRIPTION
This function sets the pose components of the sweep frame relative to the node frame. The sweep frame is the
reference frame for the curve to be swept. The sweep direction is along the z axis of this sweep frame for
CX_SWEPTT types and about the z axis for CX_SWEPTR types. The node requires 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>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 given node does not exist.</td>
</tr>
<tr>
<td>CX_ATTRIBUTE_NOT_FOUND</td>
<td>The node does not have the GEOMETRY attribute.</td>
</tr>
<tr>
<td>CX_MISMATCHED_GEOM_TYPE</td>
<td>The node is not of type CURVE, CX_SWEPTT, or CX_SWEPTR.</td>
</tr>
<tr>
<td>CX_NODE_IS_CUT_OUT</td>
<td>The 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
CxGetSweepFrame