Contents

Introduction
Chapter 1:
About This Document ...................................................................... 1-1
System Requirements ....................................................................... 1-2
Reference to Other Documentation ................................................. 1-2
Customer Service ............................................................................. 1-2

Safety
Chapter 2:
Standard Conventions ...................................................................... 2-1
General Safeguarding Tips ............................................................... 2-2
Mechanical Safety Devices ............................................................... 2-2
Installation Safety ............................................................................. 2-2
Programming Safety ........................................................................ 2-3
Operation Safety ............................................................................... 2-4
Maintenance Safety .......................................................................... 2-4

OLE Usage
Chapter 3:
MotoSim Object OLE Connection .................................................... 3-1
CreateDispatch ............................................................................. 3-1
ReleaseDispatch ........................................................................... 3-1
Robot Object .................................................................................... 3-1
Robot Tool data .............................................................................. 3-1
Robot TCP (Tool Center Point) Posture .......................................... 3-2
Pulse ............................................................................................. 3-2
Rectangle Data .............................................................................. 3-2
Angle ............................................................................................. 3-2
Robot Job Handling ......................................................................... 3-2
Robot INF Script ............................................................................. 3-2
Playback Simulation .......................................................................... 3-3
Asynchronous Simulation .............................................................. 3-3
Synchronous Simulation ................................................................. 3-3
Contents

Model Object ................................................................. 3-3
Model Tree ........................................................................ 3-3
Model file ......................................................................... 3-3
Shape Object .................................................................... 3-3
  AXIS6 model ................................................................. 3-3
Process Angle Object ....................................................... 3-4
  Use Default Frame ......................................................... 3-4
  Use Path Frame ............................................................ 3-4
  Use Near Polygon ........................................................... 3-4
  Use Cylindrical Frame .................................................... 3-4

OLE

Chapter 4:
class IMotoSim : public COleDispatchDriver .................. 4-2
void CloseCell(long nCloseOption); .......................... 4-2
long FileStore(long isStoring); .................................. 4-2
CString GetFileName(); ............................................... 4-2
void InitialUpdateWindow(); ....................................... 4-3
LPDISPATCH NewCmdGeneralOpr(); .......................... 4-3
LPDISPATCH NewCmdModelSvr(); ............................... 4-3
LPDISPATCH NewCmdRobotSvr(); .............................. 4-4
void RefreshWindow(); ................................................ 4-4
void SetFileName(LPCTSTR lpszNewValue); ................. 4-4
long ShowWindow(long nCmdShow);......................... 4-5
class ICmdAutoPlace : public COleDispatchDriver .......... 4-5
long CalcQuickCycleTime(); ......................................... 4-5
double GetCycleTime(long nType1); ............................. 4-6
CString GetInfFilePath(); ........................................... 4-6
long MakeInf2Jbi(long nValue1); ................................. 4-7
void SetCycleTime(long nType1, double dNewValue1); .... 4-7
void SetInfFilePath(LPCTSTR lpszNewValue); ............. 4-8
class ICmdCollision : public COleDispatchDriver ........ 4-8
long AddModel(long hModel1); ..................................... 4-8
long Check(); ............................................................ 4-9
long DelModel(long hModel1); ....................................... 4-9
long GetCheckAll(); ..................................................... 4-9
long GetCheckAllCombinations(); .............................. 4-10
long GetCollision(); ................................................... 4-10
long GetCount(); ......................................................... 4-11
long GetHeadPosition(); .............................................. 4-11
long GetNext(long* pPos1); ....................................... 4-12
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>long GetUpdateColorForAllShapes();</td>
<td>4-12</td>
</tr>
<tr>
<td>long IntersectionCopyAxis6</td>
<td>4-13</td>
</tr>
<tr>
<td>(long hModel1, long hModel2, long hlsAxis6Model);</td>
<td>4-13</td>
</tr>
<tr>
<td>long IntersectionGetParameter(long nIdx1);</td>
<td>4-13</td>
</tr>
<tr>
<td>double IntersectionGetReconnectDist();</td>
<td>4-14</td>
</tr>
<tr>
<td>void IntersectionSetParameter</td>
<td>4-14</td>
</tr>
<tr>
<td>(long nIdx1, long nNewValue1);</td>
<td></td>
</tr>
<tr>
<td>void IntersectionSetReconnectDist</td>
<td>4-15</td>
</tr>
<tr>
<td>(double dVal1);</td>
<td></td>
</tr>
<tr>
<td>long IntersectionUpdateLines();</td>
<td>4-15</td>
</tr>
<tr>
<td>long IsCollision(long hModel);</td>
<td>4-16</td>
</tr>
<tr>
<td>void Recalc();</td>
<td>4-16</td>
</tr>
<tr>
<td>void SetCheckAll(long nNewValue);</td>
<td>4-16</td>
</tr>
<tr>
<td>void SetCheckAllCombinations(long nNewValue1);</td>
<td>4-17</td>
</tr>
<tr>
<td>void SetUpdateColorForAllShapes</td>
<td>4-17</td>
</tr>
<tr>
<td>(long nNewValue1);</td>
<td></td>
</tr>
<tr>
<td>long UpdateColor(long nRGB);</td>
<td>4-18</td>
</tr>
<tr>
<td>class ICmdGeneralOpr: public COleDispatchDriver</td>
<td></td>
</tr>
<tr>
<td>double GetCycleTime(long nType1);</td>
<td>4-18</td>
</tr>
<tr>
<td>double GetHeartBeat(long nType1);</td>
<td>4-19</td>
</tr>
<tr>
<td>long GetHold();</td>
<td>4-19</td>
</tr>
<tr>
<td>CString GetJobName();</td>
<td>4-19</td>
</tr>
<tr>
<td>CString GetRobotName();</td>
<td>4-20</td>
</tr>
<tr>
<td>long JobReset();</td>
<td>4-20</td>
</tr>
<tr>
<td>long JobStart();</td>
<td>4-20</td>
</tr>
<tr>
<td>long JobWait(long waitTime);</td>
<td>4-21</td>
</tr>
<tr>
<td>long OnPlay(long ltype);</td>
<td>4-21</td>
</tr>
<tr>
<td>void SetCycleTime(long nType1, double dNewValue1);</td>
<td>4-22</td>
</tr>
<tr>
<td>void SetHeartBeat(long nType1, double dNewValue1);</td>
<td>4-22</td>
</tr>
<tr>
<td>void SetHold(long nNewValue);</td>
<td>4-23</td>
</tr>
<tr>
<td>void SetJobName(LPCTSTR lpszNewValue);</td>
<td>4-23</td>
</tr>
<tr>
<td>void SetRobotName(LPCTSTR lpszNewValue);</td>
<td>4-23</td>
</tr>
<tr>
<td>class ICmdMatrix: public COleDispatchDriver</td>
<td></td>
</tr>
<tr>
<td>void GetAxis6(double* pdx, double* pdy, double* pdz, double* prx, double* pry, double* prz);</td>
<td>4-24</td>
</tr>
<tr>
<td>double GetElement(long nIdxRow, long nIdxCol);</td>
<td>4-24</td>
</tr>
<tr>
<td>long GetHandle();</td>
<td>4-24</td>
</tr>
<tr>
<td>void Inverse();</td>
<td>4-25</td>
</tr>
</tbody>
</table>
void Normalize(); ................................................................. 4-25
void SetAxis6(double dx, double dy, double dz,
  double rx, double ry, double rz); ....................................... 4-25
void SetElement(long nIdxRow,
  long nIdxCol, double dNewValue); ...................................... 4-26
void Sub(long hMat1, long hMat2, long nCalType); ............... 4-26
void Trans(long hMatrix, long nPrePostReplace); ................. 4-27
class ICmdModel : public COleDispatchDriver .................. 4-27
  long CopyLine2Axis6(); .................................................... 4-27
  long DeleteAllLines(); ..................................................... 4-28
  long DeleteAllShapes(LPCTSTR strType); .......................... 4-28
  long DeleteShape(long hShape); ....................................... 4-28
  double GetAttributeDoubleByIndex(long nIdx1); ............... 4-29
  long GetAttributeLongByIndex(long nIdx1); ..................... 4-29
  CString GetFileName(); .................................................. 4-30
  long GetHandle(); ............................................................ 4-30
  long GetHideSee(); ......................................................... 4-30
  long GetMatrix(long hRefModel, long hMatrix); ................. 4-31
  CString GetModelType(); ................................................ 4-31
  CString GetName(); ........................................................ 4-31
  long GetParent(); .......................................................... 4-32
  long GetPosition(long hRefModel, double* pdPx, double* pdPy,
    double* pdPz, double* pdRx, double* pdRy, double* pdRz); .... 4-32
  long GetShapeColor(long hShape, long nToFile);.................. 4-33
  double GetShapeData(long hShape, long nRow, long nCol); .... 4-33
  long GetShapeData6(long hShape, long nRow,
  double* lpdVal1, double* lpdVal2, double* lpdVal3,
  double* lpdVal4, double* lpdVal5, double* lpdVal6); .......... 4-34
  CString GetShapeName(long hShape); ................................ 4-34
  double LineGetReconnectDistance(); ................................ 4-35
  long LineReConnect(long bIsDeleteSamePoint); .................. 4-35
  void LineSetReconnectDistance(double dValue1); ............... 4-35
  long LoadFile(); ............................................................ 4-36
  long MovParent(long nNewValue); .................................... 4-36
  long NewShape(LPCTSTR strType, long color, long rowmax); ... 4-36
  long RefreshView(); ........................................................ 4-37
  long ReLocateAllShapes(long nhModel2); ......................... 4-37
long SaveFile(); ........................................................................... 4-37
long SetAttributeDoubleByIndex
(long nIdx1, double dNewValue1); ............................................ 4-38
long SetAttributeLongByIndex
(long nIdx1, long nNewValue1); ........................................... 4-38
void SetFileName(LPCTSTR lpszNewValue); ......................... 4-39
void SetHandle(long nNewValue); ........................................ 4-39
void SetHideSee(long nNewValue); ...................................... 4-39
void SetMatrix(long hRefModel, long hMatrix, long nNewValue); ................................................ 4-40
void SetModelType(LPCTSTR lpszNewValue); .................... 4-40
void SetName(LPCTSTR lpszNewValue); .................................. 4-41
void SetParent(long nNewValue); ......................................... 4-41
void SetPosition(long hRefModel, double dPx, double dPy, double dPz, double dRx, double dRy, double dRz, long nNewValue); ........................................................................... 4-42
void SetShapeColor(long hShape, long nToFile, long nNewValue); ................................................ 4-42
void SetShapeData(long hShape, long nRow, long nCol, double dNewValue); ............................................ 4-43
void SetShapeData6(long hShape, long nRow, double dVal1, double dVal2, double dVal3, double dVal4, double dVal5, double dVal6); ............................................ 4-43
void SetShapeName(long hShape, LPCTSTR lpszNewValue); ................................................ 4-44
long ShapeCopyRow(long nhShape1, long nIdx1Start, long nhModel2, long nhShape2, long nIdx2Start, long nCount1); ............................................ 4-44
long ShapeGetCount(); .............................................................. 4-45
long ShapeGetHeadPosition(); ............................................... 4-45
long ShapeGetNext(long* pPos); ........................................ 4-45
long ShapeGetRowColMax(long hShape, long* plRow, long* plCol); .............................................................. 4-46
CString ShapeGetType(long hShape); .................................... 4-46
long ShapeMerge(long nhShape1, long nhShape2); ................. 4-47
long ShapeRefreshView(long hShape); .................................... 4-47
class ICmdModelSvr : public COleDispatchDriver......................... 4-48
long ConvertMatrix(long hModel1, long hMat1, long hModel2, long hMat2); ............................................ 4-48
long Delete(long hModel); ..................................................... 4-48
MotoSim OLE User’s Manual

Contents

long GetChildItem(long hModel1); ............................................ 4-49
long GetHandle(LPCTSTR strName); ......................................... 4-49
CString GetModelPath(); ...................................................... 4-49
CString GetName(long hModel); ........................................... 4-50
long GetNextItem(long hModel1, long nCode); ......................... 4-50
CString GetType(long hModel1); ........................................... 4-51
long New(LPCTSTR strName); .................................................. 4-51
LPDISPATCH NewCmdCollision (long hModel1, long hModel2); .... 4-51
LPDISPATCH NewCmdMatrix(); ............................................. 4-52
LPDISPATCH NewCmdMIOBitConnect(); .................................. 4-52
LPDISPATCH NewCmdModel(long hModel); ................................ 4-52
LPDISPATCH NewCmdModelIO(long hModel); ......................... 4-53
LPDISPATCH NewCmdModelType(LPCTSTR strType1); ............... 4-53
LPDISPATCH NewCmdProcessAngle(); ................................... 4-53
void Recalc(); .................................................................. 4-54
long RelocateModel(long nTag1, LPCTSTR strTypeName); ....... 4-54
void SetName(long hModel, LPCTSTR lpszNewValue); ............... 4-55
void setType(long hModel1, LPCTSTR lpszNewValue); ............. 4-55
class ICmdMotoPosition : public COleDispatchDriver .......... 4-56
long AttrGetHandle(long nTag1); ........................................... 4-56
long AttrSetHandle(long nTag1, long nhAttr1); ..................... 4-56
long JointGetDefaultAsPositionType(long nTag1); ............... 4-57
double JointGetValueDouble(long nIdx1); ......................... 4-57
long JointGetValueLong(long nIdx1); ................................... 4-57
long JointGetXyzRxyz(double* pdVal1, double* pdVal2, double* pdVal3, double* pdVal4, double* pdVal5, double* pdVal6); .................. 4-58
long JointSetValueDouble (long nIdx1, double dNewValue1); .... 4-58
long JointSetValueLong(long nIdx1, long nNewValue1); .......... 4-59
long JointSetXyzRxyz(double dVal1, double dVal2, double dVal3, double dVal4, double dVal5, double dVal6); ...... 4-59
long PrpGetPositionType(); .................................................. 4-60
long PrpSetPositionType(long nNewValue); ........................... 4-60
class ICmdMotoRobot : public COleDispatchDriver .......... 4-61
long AttrGetCount(long nTag1); ........................................... 4-61
long AttrGetDefaultRect(LPDISPATCH lpdispatch); .............. 4-61
long AttrGetDefaultRect(LPDISPATCH lpdispatch); .............. 4-61
long AttrGetHeadPosition(long nTag1); .................................... 4-62
CString AttrGetItemText(long nTag1, long nAttr1); .................. 4-62
long AttrGetNext(long nTag1, long nPos1, long* pnPos2); ....... 4-63
long AttrSetDefaultRect(LPDISPATCH lpDispatch); ................... 4-63
long GetCurrentJobLine(); ......................................................... 4-64
CString GetCurrentJobName(); .................................................. 4-64
long GetCurrentJobStep(); ......................................................... 4-64
long GetCurrentToolHandle(); .................................................. 4-65
double GetCycleTime(long nType1); ........................................ 4-65
long GetHandle(); ...................................................................... 4-65
CString GetMdlFilePath(); .......................................................... 4-66
long GetMotionModuleCode(); ................................................. 4-66
long GetPowerStatus(long nType1); .......................................... 4-66
CString GetRobotPath();............................................................. 4-67
JobGetHeadPosition(); ....................................................... 4-67
CString JobGetNext(long* pPos); .............................................. 4-67
MpsConvertPosition(long nhRbModule, LPDISPATCH lpdispatch1, LPDISPATCH lpdispatch2); 4-68
MpsGetPositionCur(long nhRbModule, LPDISPATCH lpdispatch); .......................................................... 4-69
MpsGetPositionCurTcpSymmetry(long nhRbModule, LPDISPATCH lpdispatch); .......................................................... 4-69
LPDISPATCH NewCmdMotoJob(LPCTSTR strJobName); .......... 4-70
LPDISPATCH NewCmdMotoRobotMemory(); ........................... 4-70
LPDISPATCH NewCmdMotoRobotSystem(); ............................. 4-70
long RbModuleGetCount(); ....................................................... 4-71
long RbModuleGetHeadPosition(); ............................................ 4-71
CString RbModuleGetItemText(long nhRbModule); ................. 4-71
long RbModuleGetJointCount(long nhRbModule); .................. 4-72
long RbModuleGetNext(long* pnPos1); .................................... 4-72
long RbModuleJointGetAxisBits(long nhRbModule); ................ 4-72
void SetCurrentJobLine(long nNewValue); ............................... 4-73
void SetCurrentJobName(LPCTSTR lpszNewValue); ................. 4-73
void SetCurrentToolHandle(long nTool1); ............................... 4-73
void SetCycleTime(long nType1, double dNewValue1); ........... 4-74
void SetMdlFilePath(LPCTSTR lpszNewValue); .......................... 4-74
void SetMotionModuleCode(long nNewValue); ........................ 4-75
void SetPowerStatus(long nType1, long nNewValue1); .......... 4-75
class ICmdProcessAngle : public COleDispatchDriver ................... 4-76
long AdjustAfterMulMatrix ......................................................... 4-76
(long nPos1, long nIdx1, double dNewValue1); ....................... 4-76
long AdjustAfterMulMatrixAll .................................................. 4-76
(long nIdx1, double dNewValue1); ........................................... 4-76
long AdjustGetCondition(long nIdx1); .................................. 4-77
long AdjustPathAllAsCylindricalFrame .................................. 4-77
(long hModel1, long hModel2); .............................................. 4-77
long AdjustPathAllAsDefaultFrame .......................................... 4-78
(long hModel1, long hModel2); .............................................. 4-78
long AdjustPathAllAsMirrorFrame ........................................... 4-79
(long hModel1, long hModel2, long nNewVal1Face, 
long nNewVal2Pose); ............................................................. 4-79
long AdjustPathAllAsNearPolygonFrame ................................. 4-80
(long hModel1, long hModel2, double dDistance, 
long boolRecursive); ............................................................ 4-80
long AdjustPathAllAsNextPoint(long nNewAxisDir); ............... 4-81
long AdjustPathAllAsPathFrame(); ........................................... 4-81
long AdjustSetCondition(long nIdx1, long nNewVal1); .......... 4-82
long Axis6AddTail(); .............................................................. 4-82
long Axis6InsertAfter(long nPos1); ....................................... 4-83
long Axis6InsertBefore(long nPos1); .................................... 4-83
void Axis6RemoveAll(); ......................................................... 4-83
void Axis6RemoveAt(long nPos1); .......................................... 4-84
long GetCountCol(); .............................................................. 4-84
long GetCountRow(); .............................................................. 4-84
double GetData(long nhAxis6, long nIdx1); .......................... 4-85
long GetHeadPosition(); ......................................................... 4-85
long GetNext(long* nPos1); .................................................... 4-85
void SetData(long nhAxis6, 
long nIdx1, double dNewValue1); ........................................ 4-86
long UpdateAxis6(long hModel1, 
long hShape1Axis6, long nIsUpdate); .................................. 4-86
class ICmdRobotSvr : public ColeDispatchDriver ........................................ 4-87
  long DeleteRobot(long hRobot); .................................................. 4-87
  long GetCount(); ..................................................................... 4-87
  long GetRobotHeadPosition(); ................................................... 4-87
  CString GetRobotName(long hRobot); ...................................... 4-88
  long GetRobotNext(long* pPos); ............................................. 4-88
  LPDISPATCH NewCmdAutoPlace(long hRobot); ......................... 4-88
  LPDISPATCH NewCmdMotoPosition(); ..................................... 4-89
  LPDISPATCH NewCmdMotoRobot(long hRobot); ............................ 4-89
  LPDISPATCH NewCmdMotoRobotTool(); .................................... 4-89
  long NewRobot(LPCTSTR strName, LPCTSTR strRobotPath); .......... 4-90
  void SetRobotName(long hRobot, LPCTSTR lpszNewValue); .......... 4-90

class ICmdModelIO : public COleDispatchDriver .................................. 4-91
  long Create(long nType); .......................................................... 4-91
  void Delete(long hMSimIO); ..................................................... 4-91
  long GetBit(long hMdlIo, long nIdx1); ...................................... 4-92
  long GetByte(long hMdlIo, long nIdx1); ................................... 4-92
  long GetCount(); ..................................................................... 4-93
  long GetHeadPosition(); ............................................................ 4-93
  CString GetItemText(long hMdlIo); ........................................... 4-93
  long GetMaxNum(long hMdlIo);................................................... 4-94
  long GetNext(long* pnPos1);..................................................... 4-94
  long GetNumLinkFrom(long hMSimIO, long nIdx1); ............... 4-94
  long GetNumLinkTo(long hMSimIO, long nIdx1); .................... 4-95
  long GetType(long hMdlIo); ...................................................... 4-95
  long IsExist(); ........................................................................ 4-95
  void SetBit(long hMdlIo, long nIdx1, long nNewValue); .......... 4-96
  void SetByte(long hMdlIo, long nIdx1, long nNewValue); ........ 4-96
  void SetItemText(long hMdlIo, LPCTSTR lpszNewValue); ........ 4-97
  void SetMaxNum(long hMdlIo, long nNewValue); .................... 4-97
  void SetType(long hMdlIo, long nNewValue); ........................... 4-98
class ICmdMIOBitConnect : public COleDispatchDriver .................. 4-99
  long Add(long hModel1, long hSimIO1, long nIdx1,
            long hModel2, long hSimIO2, long nIdx2); ................. 4-99
  long Delete(long hModel1, long hSimIO1,
              long nIdx1, long hModel2, long hSimIO2, long nIdx2); ... 4-100
  long DeleteAll(); .......................................................... 4-100
  long Node1GetCount(); .................................................... 4-101
  long Node1GetHeadPosition(); ........................................... 4-101
  long Node1GetSimIOHandle(long hNode1); ......................... 4-101
  long Node1GetSimIOHandle(long hNode1); ...................... 4-102
  long Node1GetNext(long* pnPos1); .................................. 4-102
  long Node2GetCount(long hNode1); .................................. 4-103
  long Node2GetHeadPosition(long hNode1); ....................... 4-103
  long Node2GetIndex1(long hNode2); ............................... 4-103
  long Node2GetIndex2(long hNode2); .................................. 4-103
  long Node2GetModelHandle(long hNode2); ....................... 4-104
  long Node2GetSimIOHandle(long hNode2); ...................... 4-104
  long Node2GetNext(long hNode1, long* pnPos1); .............. 4-104
  long OnUpdate(); .......................................................... 4-105
  long OnUpdateAll(); ....................................................... 4-105
class ICmdMotoRobotMemory : public COleDispatchDriver ........... 4-106
  long FileStoring(LPCTSTR strFile, long fIsStoring); .......... 4-106
  long ToolGetCount(); .................................................... 4-106
  long ToolGetHeadPosition(); .......................................... 4-106
  long ToolGetNext(long* pnPos1); .................................. 4-107
  long ToolGetProperty(long nTool1, LPDISPATCH lpdispatch); ... 4-107
  long ToolSetProperty((long nTool1, LPDISPATCH lpdispatch);... 4-108
class ICmdMotoRobotTool : public COleDispatchDriver ............... 4-109
  CString GetItemText(); ................................................... 4-109
  double GetMass(); .......................................................... 4-109
  long GetToolNumber(); .................................................... 4-110
  double GetXyzMxyzOfMass(long nIdx); ............................ 4-110
  double GetXyzRxyz(long nIdx); ...................................... 4-110
  void SetItemText(LPCTSTR strNewValue); ......................... 4-111
  void SetMass(double dNewValue); .................................. 4-111
  void SetToolNumber(long nNewValue); ............................. 4-111
  void SetXyzMxyzOfMass(long nIdx, double dNewValue); ....... 4-111
  void SetXyzRxyz(long nIdx, double dNewValue); .................. 4-112
class ICmdMotoRobotSystem : public COleDispatchDriver ........ 4-113
long GetJointMaxPulse(long nIdx1);................................. 4-113
long GetJointMinPulse(long nIdx1); .................................. 4-113
long GetJointType(long nIdx1); ....................................... 4-114
long GetJointYaXb(long nIdx1, double* pdA, double* pdB);; 4-114
void SetJointMaxPulse(long nIdx1, long nNewValue1);......... 4-115
void SetJointMinPulse(long nIdx1, long nNewValue1);......... 4-115
void SetJointType(long nIdx1, long nNewValue1); ............. 4-116
void SetJointYaXb(long nIdx1, double
dNewValue1, double dNewValue2); ........................................ 4-116

class ICmdModelType : public COleDispatchDriver................ 4-117
long GetCount(); .......................................................... 4-117
long GetHeadPosition(); ............................................... 4-117
CString GetItemText(); .................................................. 4-117
long GetNext(long* pPos1); ......................................... 4-118

Index
Chapter 1: Introduction

MotoSim supports OLE (Object Linking and Embedding) Automation. OLE Automation allows other PC-based application packages to control MotoSim. Users can, for example, set serial communications port settings, turn on DCI, or reset the communications buffer from other software applications.

OLE, or ActiveX as it has become known, is a vast standard that has had literally hundreds of books dedicated to explaining it. Fortunately, creating and using an OLE client in Visual Basic is quite simple.

OLE supports Distributed Communications (DCOM) which enables users to control Visual DCI from remote computers linked together via Ethernet. This powerful feature allows limited robot control and monitoring from a remote PC.

Visual DCI is set up to run a new copy each time a new OLE Client requests a Visual DCI session. This allows simultaneous communication between Visual DCI and multiple robot controllers. You can run as many copies of Visual DCI as you have serial ports on your computer.

About This Document

Please read this manual for safety precautions, installation instructions, and other important information before using MotoSim OLE. This manual is intended as an introduction and overview for personnel who are familiar with the operation of their Motoman robot model, and Microsoft® Windows®/ PC usage. This manual contains the following chapters:

Chapter 1: Introduction
Provides general information about MotoSim OLE Automation, a list of reference documents, and customer service information.

Chapter 2: Safety
Provides information regarding the safe use and operation of MotoSim OLE Automation.

Chapter 3: OLE Usage
Provides instructions for Visual DCI 4 software installation, hardware key installation, and first time use.

Chapter 4: OLE Reference
Provides instructions for Visual DCI 4 software installation, hardware key installation, and first time use.
1-2 Introduction

**System Requirements**

Computer - IBM PC or compatible  
Processor - Intel Pentium 200  
Memory - 8 MB  
Hard Disk - 6 MB minimum available  
CD ROM Drive  
Monitor - VGA Graphics  
COM Port - One RS-232 port capable of operating at 9600 baud for each copy of MotoSim.  
Interface Cable - One RS-232 serial interface cable for each copy of MotoSim (Supplied with MotoSim)  
Operating System - MicroSoft Windows 98, NT or Windows XP  
Controller Software - Controller software must have the DCI Communications option installed.

**Reference to Other Documentation**

For additional information, refer to the following:  
• Operator’s Manual for your application  
• Manipulator Manual for your robot model  
• MotoSim User’s Manual

**Customer Service**

If you need technical assistance, contact the Motoman service staff at (937) 847-3200. Please have the following information ready before you call:  
• MotoSim version  
• Operating system (Microsoft Windows 98/NT/2000/XP)  
• System configuration (hard disk capacity, memory, software, etc.)  
• List of all software installed after shipment from Motoman  
• Description of difficulty (make note of any error messages)  
• Application specific information including robot jobs, process details etc...
Chapter 2: Safety

It is the purchaser’s responsibility to ensure that all local, county, state, and national codes, regulations, rules, or laws relating to safety and safe operating conditions for each installation are met and followed.

We suggest that you obtain and review a copy of the ANSI/RIA National Safety Standard for Industrial Robots and Robot Systems. This information can be obtained from the Robotic Industries Association by requesting ANSI/RIA R15.06. The address is as follows:

Robotic Industries Association
900 Victors Way
P.O. Box 3724
Ann Arbor, Michigan 48106
TEL: (734) 994-6088
FAX: (734) 994-3338

Ultimately, the best safeguard is trained personnel. The user is responsible for providing personnel who are adequately trained to operate, program, and maintain the robot cell. The robot must not be operated by personnel who have not been trained!

We recommend that all personnel who intend to operate, program, repair, or use the robot system be trained in an approved Motoman training course and become familiar with the proper operation of the system.

This safety section addresses the following:

- Standard Conventions
- General Safeguarding Tips
- Mechanical Safety Devices
- Installation Safety
- Programming Safety
- Operation Safety
- Maintenance Safety

Standard Conventions

This manual includes information essential to the safety of personnel and equipment. As you read through this manual, be alert to the four signal words:

- DANGER
- WARNING
- CAUTION
- NOTE

Pay particular attention to the information provided under these headings which are defined below (in descending order of severity).

⚠️ DANGER!

Information appearing under the DANGER! caption concerns the protection of personnel from the immediate and imminent hazards that, if not avoided, will result in immediate, serious personal injury or loss of life in addition to equipment damage.

⚠️ WARNING!

Information appearing under the WARNING! caption concerns the protection of personnel and equipment from potential hazards that can result in personal injury or loss of life in addition to equipment damage.
Information appearing under the CAUTION! caption concerns the protection of personnel, equipment, software, and data from hazards that can result in minor personal injury or equipment damage.

NOTE: Information appearing in a NOTE caption provides additional information which is helpful in understanding the item being explained.

General Safeguarding Tips
All operators, programmers, plant and tooling engineers, maintenance personnel, supervisors, and anyone working near the robot must become familiar with the operation of this equipment. All personnel involved with the operation of the equipment must understand potential dangers of operation. General safeguarding tips are as follows:

- Improper operation can result in personal injury and/or damage to the equipment. Only trained personnel familiar with the operation of this robot, the operator's manuals, the system equipment, and options and accessories should be permitted to operate this robot system.
- Do not enter the robot cell while it is in automatic operation. Programmers must have the teach pendant when they enter the robot cell.
- Improper connections can damage the robot. All connections must be made within the standard voltage and current ratings of the robot I/O (Inputs and Outputs).
- The robot must be placed in Emergency Stop (E-STOP) mode whenever it is not in use.
- In accordance with ANSI/RIA R15.06, section 6.13.4 and 6.13.5, use lockout/tagout procedures during equipment maintenance. Refer also to Section 1910.147 (29CFR, Part 1910), Occupational Safety and Health Standards for General Industry (OSHA).

Mechanical Safety Devices
The safe operation of the robot, positioner, auxiliary equipment, and system is ultimately the user's responsibility. The conditions under which the equipment will be operated safely should be reviewed by the user. The user must be aware of the various national codes, ANSI/RIA R15.06 safety standards, and other local codes that may pertain to the installation and use of industrial equipment. Additional safety measures for personnel and equipment may be required depending on system installation, operation, and/or location. The following safety measures are available:

- Safety fences and barriers
- Light curtains
- Door interlocks
- Safety mats
- Floor markings
- Warning lights

Check all safety equipment frequently for proper operation. Repair or replace any non-functioning safety equipment immediately.

Installation Safety
Safe installation is essential for protection of people and equipment. The following suggestions are intended to supplement, but not replace, existing federal, local, and state laws and regulations. Additional safety measures for personnel and equipment may be required depending on system installation, operation, and/or location. Installation tips are as follows:

- Be sure that only qualified personnel familiar with national codes, local codes, and ANSI/RIA R15.06 safety standards are permitted to install the equipment.
- Identify the work envelope of each robot with floor markings, signs, and barriers.
• Position all controllers outside the robot work envelope.
• Whenever possible, install safety fences to protect against unauthorized entry into the work envelope.
• Eliminate areas where personnel might get trapped between a moving robot and other equipment (pinch points).
• Provide sufficient room inside the workcell to permit safe teaching and maintenance procedures.

Programming Safety
All operators, programmers, plant and tooling engineers, maintenance personnel, supervisors, and anyone working near the robot must become familiar with the operation of this equipment. All personnel involved with the operation of the equipment must understand potential dangers of operation. Programming tips are as follows:

• Any modifications to PART 1 of the controller PLC can cause severe personal injury or death, as well as damage to the robot! Do not make any modifications to PART 1. Making any changes without the written permission of Motoman will VOID YOUR WARRANTY!
• Some operations require standard passwords and some require special passwords. Special passwords are for Motoman use only. YOUR WARRANTY WILL BE VOID if you use these special passwords.
• Back up all programs and jobs onto a floppy disk whenever program changes are made. To avoid loss of information, programs, or jobs, a backup must always be made before any service procedures are done and before any changes are made to options, accessories, or equipment.
• The concurrent I/O (Input and Output) function allows the customer to modify the internal ladder inputs and outputs for maximum robot performance. Great care must be taken when making these modifications. Double-check all modifications under every mode of robot operation to ensure that you have not created hazards or dangerous situations that may damage the robot or other parts of the system.
• Improper operation can result in personal injury and/or damage to the equipment. Only trained personnel familiar with the operation, manuals, electrical design, and equipment interconnections of this robot should be permitted to operate the system.
• Inspect the robot and work envelope to be sure no potentially hazardous conditions exist. Be sure the area is clean and free of water, oil, debris, etc.
• Be sure that all safeguards are in place.
• Check the E-STOP button on the teach pendant for proper operation before programming.
• Carry the teach pendant with you when you enter the workcell.
• Be sure that only the person holding the teach pendant enters the workcell.
• Test any new or modified program at low speed for at least one full cycle.
• Incorrect use of MotoCal for robot calibration may cause incorrect robot calibration resulting in robot crash and/or personal injury or death.
• Incorrect use of MotoCal for job Filtering or Downloading, may cause robot crash and/or personal injury or death.
Operation Safety

All operators, programmers, plant and tooling engineers, maintenance personnel, supervisors, and anyone working near the robot must become familiar with the operation of this equipment. All personnel involved with the operation of the equipment must understand potential dangers of operation. Operation tips are as follows:

- Be sure that only trained personnel familiar with the operation of this robot, the operator's manuals, the system equipment, and options and accessories are permitted to operate this robot system.
- Check all safety equipment for proper operation. Repair or replace any non-functioning safety equipment immediately.
- Inspect the robot and work envelope to ensure no potentially hazardous conditions exist. Be sure the area is clean and free of water, oil, debris, etc.
- Ensure that all safeguards are in place.
- Improper operation can result in personal injury and/or damage to the equipment. Only trained personnel familiar with the operation, manuals, electrical design, and equipment interconnections of this robot should be permitted to operate the system.
- Do not enter the robot cell while it is in automatic operation. Programmers must have the teach pendant when they enter the cell.
- The robot must be placed in Emergency Stop (E-STOP) mode whenever it is not in use.
- This equipment has multiple sources of electrical supply. Electrical interconnections are made between the controller, external servo box, and other equipment. Disconnect and lockout/tagout all electrical circuits before making any modifications or connections.
- All modifications made to the controller will change the way the robot operates and can cause severe personal injury or death, as well as damage the robot. This includes controller parameters, ladder parts 1 and 2, and I/O (Input and Output) modifications. Check and test all changes at slow speed.

Maintenance Safety

All operators, programmers, plant and tooling engineers, maintenance personnel, supervisors, and anyone working near the robot must become familiar with the operation of this equipment. All personnel involved with the operation of the equipment must understand potential dangers of operation. Maintenance tips are as follows:

- Do not perform any maintenance procedures before reading and understanding the proper procedures in the appropriate manual.
- Check all safety equipment for proper operation. Repair or replace any non-functioning safety equipment immediately.
- Improper operation can result in personal injury and/or damage to the equipment. Only trained personnel familiar with the operation, manuals, electrical design, and equipment interconnections of this robot should be permitted to operate the system.
- Back up all your programs and jobs onto a floppy disk whenever program changes are made. A backup must always be made before any servicing or changes are made to options, accessories, or equipment to avoid loss of information, programs, or jobs.
- Do not enter the robot cell while it is in automatic operation. Programmers must have the teach pendant when they enter the cell.
- The robot must be placed in Emergency Stop (E-STOP) mode whenever it is not in use.
- Be sure all safeguards are in place.
- Use proper replacement parts.
- This equipment has multiple sources of electrical supply. Electrical interconnections are made between the controller, external servo box, and other equipment. Disconnect and lockout/tagout all electrical circuits before making any modifications or connections.
• All modifications made to the controller will change the way the robot operates and can cause severe personal injury or death, as well as damage the robot. This includes controller parameters, ladder parts 1 and 2, and I/O (Input and Output) modifications. Check and test all changes at slow speed.

• Improper connections can damage the robot. All connections must be made within the standard voltage and current ratings of the robot I/O (Inputs and Outputs).
NOTES
Chapter 3: OLE Usage

This chapter provides an overview of the main concepts of MotoSim OLE including:

- Connection
- Robot Object
- Robot Tool data
- Robot TCP Posture
- Robot Job Handling
- Robot INF Script
- Playback Simulation
- Model Object
- Shape Object
- Process Angle Object

**MotoSim Object OLE Connection**

CreateDispatch

CreateDispatch creates a MotoSim object with an identical name "MotoSim.Document".

```c
mMSim = mMotoSim.CreateDispatch(T("MotoSim.Document"));
```

mMSim is created as an IMotoSim object as shown below.

```c
IMotoSim mMotoSim;
```

The following calls are required after CreateDispatch.

```c
mMotoSim.InitialUpdateWindow();
mMotoSim.SetFileName(mstrMclPathName);
mMotoSim.FileStore(FALSE);
mMotoSim.ShowWindow(SW_SHOWNORMAL);
mMotoSim.RefreshWindow();
```

- InitialUpdateWindow initializes the MotoSim view.
- SetFileName sets the name of the MotoSim cell file (*.mcl).
- FileStore(FALSE) loads the mcl file.
- RefreshWindow refreshes the MotoSim view.

**ReleaseDispatch**

ReleaseDispatch releases the MotoSim object at the end of the process. This is required when the MotoSim object is no longer in use.

The following is the standard way to do this.

```c
mMSim.CloseCell(OLECLOSE_NOSAVE);
mMSim.ReleaseDispatch();
```

**Robot Object**

The ICmdMotoRobot Object allows applications to handle basic Robot operations. This Object also supports functions such as converting data between Rectangular and Joint data.

The ICmdRobotSvr manages Robot objects. ICmdRobotSvr provides methods to create a Robot object from Robot handle. A Robot handle can be acquired by calling GetRobotNext().

There is one Robot handle for each robot. A Robot handle is valid while the cell is opened.

**Robot Tool data**

SetCurrentToolHandle() changes the current tool handle. A tool handle is acquired using an ICmdMotoRobotMemory object. This object is dispatched from an ICmdMotoRobot object.

There is one Tool handle for each Tool. A Tool handle is valid while the cell is opened.

The ICmdMotoRobotTool object handles Tool properties.
**Robot TCP (Tool Center Point) Posture**

The Robot TCP posture is defined by Pulse data or Rectangular data. Pulse data means joint motor location or joint angle. Rectangular data means 6 element data X, Y, Z, Rx, Ry, Rz.

`MpsGetPositionCur()` saves the Robot TCP posture to an `ICmdMotoPosition` object.

```cpp
mRobot.MpsGetPositionCur(hRbModule, iMotoPosition1);
```

A `hRbModule` is robot module handle for a Robot. The robot module can be acquired by using the `RbModuleGetNext()` function.

An `iMotoPosition1` is an `ICmdMotoPosition` object created from an `ICmdRobotSvr` object.

```cpp
iMotoPosition1.AttachDispatch(mRobot.Srv.NewCmdMotoPosition());
```

**Pulse**

`PrpSetPositionType(FALSE)` changes Position data type to pulse data type.

```cpp
iMotoPosition1.PrpSetPositionType(FALSE);
```

Use the `JointGetValueLong()` function to retrieve data.

```cpp
for (i=0;i<nCount1;i++)nPulse1[i]=iMotoPosition1.JointGetValueLong(i);
```

**Rectangle Data**

`PrpSetPositionType(TRUE)` changes Position data type to rectangular data type.

```cpp
iMotoPosition1.PrpSetPositionType(TRUE);
```

Use the `JointGetXyzRxyz()` function to retrieve data.

```cpp
iMotoPosition1.JointGetXyzRxyz(&dx, &dy, &dz, &dxr, &dry, &drz);
```

These values are related Robot frame.

**Angle**

`PrpSetPositionType(2)` changes Position data type to angle data type.

```cpp
iMotoPosition1.PrpSetPositionType(2);
```

Use the `JointGetValueDouble()` function to retrieve data.

```cpp
for (i=0;i<nCount1;i++)dValue[i]=iMotoPosition1.JointGetValueDouble(i);
```

**Robot Job Handling**

The `ICmdMotoRobot` object provides Job handling.

- `SetCurrentJobName()` selects current job.
- `void SetCurrentJobName (LPCSTR lpszNewValue)`;
- `SetCurrentJobLine()` changes the current job line.
- `void SetCurrentJobLine (nLine)`;

The Job file path is the same as the Robot path. The Robot path can be acquired by calling `GetRobotPath()`.

**Robot INF Script**

The purpose of an INF script is to program both a Robot motion and a Model operation. Robot motion can be defined using a model-shape index. Model operations can be programmed using the MODELDRV command. The MODELDRV command generates MDL commands in Job.

A Robot motion can be taught as shown in the following script.

```cpp
TBOXTEACH=(MOV1, ModelName:AXIS6[0] [1], NULL, NULL);
```

The script shows that the TCP goes to an AXIS6 shape index location. Thus the script does not specify Robot joint pulses. This allows scripting a Robot path based on the model.
Playback Simulation

The ICmdGeneralOpr object provides playback operation. There are two ways of simulating Robot motion; Asynchronous and Synchronous.

Asynchronous Simulation

The JobStart() command starts asynchronous simulation. A simulation will continue until the Robot reaches the end of Job. The JobReset() is required before calling JobStart().

The SetHold() can be called for holding simulation, if necessary.

Synchronous Simulation

The OnPlay() command provides a synchronous simulation. The following call sequences are the typical simulation program.

OnPlay (0) ;// Job Reset
OnPlay (2) ;// Job Play
While (OnPlay (3)==TRUE) ;// Simulation until reaching Job end
OnPlay (1) ;// Job Stop

Model Tree

The ICmdModelSvr provides New/Delete model functions. The New() function returns model handle if succeed. The NewCmdModel() function creates an ICmdModel object. The ICmdModel object provides the SetPosition() function to change the model location. The Model tree has parent name, model file, and location data. The Model is element of robot arms. Robot arms are implemented with using the Model structure.

Model file

A Model file is either a "*.mdl" file or a "*.rwx" file. A "*.mdl" file is the MotoSim primitive model-file format. A "*.rwx" file is a RenderWare format model-file.

Use SetFileName()/ GetFileName() to set or get a file name.

Use LoadFile()/ SaveFile() to load or save a file. The LoadFile() works with "*.mdl" files and "*.rwx" files. The SaveFile() works with "*.mdl" files.

Shape Object

Shape is an element of a model. Shape has defined types, for example BOX, CYLINDER, and AXIS6. The NewShape() creates new a Shape and returns a Shape handle.

ShapeGetHeadPosition()/ ShapeGetNext() get a handle of each Shape in model.

GetShapeData6()/ SetShapeData6() provide access to Shape data. Shape data is formatted as a Row and Column array.

AXIS6 model

The AXIS6 Shape type has 6 elements that are X, Y, Z, Rx, Ry, Rz,

The Data type is named Roll Pitch Yaw. The ICmdMatrix provides the method to convert between AXIS6 data to a matrix.
**Process Angle Object**

The ICmdProcessAngle object provides methods to create AXIS6 Shapes from line Shapes. The converting methods are "Use Default Frame", "Use Path Frame", "Use Near Polygon", and "Use Cylindrical Frame".

**Use Default Frame**

Rx, Ry, and Rz can be set to be the same as the selected model frame.

**Use Path Frame**

Rx, Ry, and Rz can be set to be the same as the Path frame. The Path frame is formed from the nearest three indexed points around each point.

If (N) points are formed as one path, the Point indexes will be P(0), P(1), P(2), ... and P(N-1).

The indexed point P(n) will have a frame formed with P(n), P(n+1), and P(n+2). P(n-1) would be taken instead of P(n+2), if P(n+2) exceed max number.

If P(n) is the last point, the indexed point P(n) will have the same frame as the P(n-1) frame.

**Use Near Polygon**

Rx, Ry, and Rz can be set the same as the nearest polygon frame. The Rz direction would be taken as perpendicular to the polygon face.

**Use Cylindrical Frame**

Rx, Ry, and Rz can be set the same as the cylindrical frame. The cylindrical frame is formed with a centered model frame, which is selectable.

A Line Shape can be created from the Intersection Line of an ICmdCollision object. An ICmdCollision object provides method to check for collision between two or more models. This object provides methods for making intersection lines also.

The intersection line is a LINE shape that has X, Y, Z data.

Converting AXIS6 shape may be required to use these lines as teaching position.
MotoSim supplies MotoSim.tlb file to update the interface specification.

The MotoSim OLE Automation object has separated categorized classes. These object classes are designed to work together.

```c++
class IMotoSim : public COleDispatchDriver
class ICmdAutoPlace : public COleDispatchDriver
class ICmdCollision : public COleDispatchDriver
class ICmdGeneralOpr : public COleDispatchDriver
class ICmdMatrix : public COleDispatchDriver
class ICmdModel : public COleDispatchDriver
class ICmdModelSvr : public COleDispatchDriver
class ICmdMotoPosition : public COleDispatchDriver
class ICmdMotoRobot : public COleDispatchDriver
class ICmdProcessAngle : public COleDispatchDriver
class ICmdRobotSvr : public COleDispatchDriver
class ICmdModelIO : public COleDispatchDriver
class ICmdMIOBitConnect : public COleDispatchDriver
class ICmdMotoRobotMemory : public COleDispatchDriver
class ICmdMotoRobotTool : public COleDispatchDriver
class ICmdMotoRobotSystem : public COleDispatchDriver
class ICmdModelType : public COleDispatchDriver
```

⚠️ CAUTION!

The function specification can be changed without notice.
class IMotoSim : public COleDispatchDriver
void CloseCell(long nCloseOption);

Description
Call this member function to close cell.
Parameters
long nCloseOption is required.
Return Value
None
See Also
FileStore()
InitialUpdateWindow()

long FileStore(long isStoring);

Description
Call this member function to load or save cell.
Parameters
long isStoring
TRUE (1) for saving cell file.
FALSE (0) for loading cell file.
Return Value
TRUE (1) if succeed.
FALSE (0) if failed.
See Also
SetFileName()
GetFileName()
CString GetFileName();

Description
GetFileName function gets the cell file path.

Parameters
None

Return Value
Cell file path (... \... \*.mcl).

See Also
SetFileName()
FileStore()

void InitialUpdateWindow();

Description
Call this member function once after creating MotoSim object.

Parameters
None

Return Value
None

See Also
LPDISPATCH NewCmdGeneralOpr();

Description
NewCmdGeneralOpr() function creates ICmdGeneralOpr object.

Parameters
None

Return Value
LPDISPATCH
The dispatch handle

See Also
ICmdGeneralOpr object.
LPDISPATCH NewCmdModelSvr();

**Description**
NewCmdModelSvr() function creates ICmdModelSvr object.

**Parameters**
None

**Return Value**
LPDISPATCH
The dispatch handle

**See Also**
ICmdModelSvr object.

LPDISPATCH NewCmdRobotSvr();

**Description**
NewCmdRobotSvr() function creates ICmdRobotSvr object.

**Parameters**
None

**Return Value**
LPDISPATCH
The dispatch handle

**See Also**
ICmdRobotSvr object.

void RefreshWindow();

**Description**
Call this member function to update MotoSim views.

**Parameters**
None

**Return Value**
None

**See Also**
void SetFileName(LPCTSTR lpszNewValue);

**Description**
SetFileName function sets the cell file path.

**Parameters**
Cell file path (... \.. \*.mcl).

**Return Value**
None

**See Also**
GetFileName()
FileStore()

long ShowWindow(long nCmdShow);

**Description**
The ShowWindow function sets the specified window's show state.

**Parameters**
nCmdShowA long value data for Window Style.
SW_SHOWNORMALActivates and displays a window.
For more information, see ShowWindow function in Window help.

**Return Value**
If the window was previously visible, the return value is nonzero.

**See Also**
class ICmdAutoPlace : public COleDispatchDriver
long CalcQuickCycleTime();

**Description**
Simulates selected Inf-file with silent mode. The simulation graphics does not update its screen while simulating, but cycle time is reported.

**Parameters**
None

**Return Value**
TRUE (1) if succeed
FALSE (0) if failed

**See Also**
MakeInf2Jbi();
GetCycleTime();
SetCycleTime();

double GetCycleTime(long nType1);

**Description**
Gets cycle time (sec)

**Parameters**
long nType1 indicate type of time.
1: Returns Move time
2: Returns Play time

**Return Value**
Cycle time (sec)

**See Also**
MakeInf2Jbi();
SetCycleTime()

CString GetInfFilePath();

**Description**
Gets Inf file path

**Parameters**
None

**Return Value**
Inf file path

**See Also**
SetInfFilePath();
MakeInf2Jbi();
long MakeInf2Jbi(long nValue1);

**Description**
Converts inf file to jbi file. There could be an error while converting Robot position. A *.log file would be created if an error occurs.

**Parameters**
long nValue1
Inf file path.

**Return Value**
TRUE (1) if succeed
FALSE (0) if failed. A *.log file will be created for reporting error.

**See Also**
SetInfFilePath();
GetInfFilePath();

void SetCycleTime(long nType1, double dNewValue1);

**Description**
Initializes cycle time value.

**Parameters**
long nType1 indicate type of time.
1: For setting Move time
2: For setting Play time
double dNewValue1 new value.

**Return Value**
None

**See Also**
MakeInf2Jbi();
GetCycleTime()
void SetInfFilePath(LPCTSTR lpszNewValue);

**Description**
Sets Inf file path

**Parameters**
LPCTSTR lpszNewValue
Inf file path

**Return Value**
None

**See Also**
GetInfFilePath();
MakeInf2Jbi();

class ICmdCollision : public COleDispatchDriver

long AddModel(long hModel1);

**Description**
Adds the model to the list to be checked for collisions.

**Parameters**
long hModel1Model handle

**Return Value**
Number of models in the list for collision detection.

**See Also**
DelModel()
GetHeadPosition()
GetNext()
GetCount()
long Check();

Description
Check whether a collision occurred.
Parameters
None
Return Value
TRUE (1) if collision
FALSE (0) if no collision
See Also
AddModel();
DelModel();

long DelModel(long hModel1);

Description
Delete model in collision list.
Parameters
long hModel1: Model handle to be deleted.
Return Value
TRUE: if delete is succeed.
FALSE: if delete is failed.
See Also
AddModel()

long GetCheckAll();

Description
Gets the flag status of "CheckAll". This flag changes calculation cycle for collision. The calculation can be
stopped, if the first collision shape is found.
Parameters
None
Return Value
TRUE (1) or FALSE (0)
See Also
SetCheckAll()
long GetCheckAllCombinations();

Description
Gets the status of the "nIsAllCombinations" flag. This flag controls the combinations of models checked for collision. Consider the case of three models added to "ICmdCollision" object, model A, model B, and model C. Model A is the first model to be added. If the "nIsAllCombinations" flag is FALSE(0) the combinations of A-B and A-C are considered for collision calculation. The combinations of A-B, A-C, and B-C are considered for collision calculation, if the "nIsAllCombinations" flag is TRUE(0). By setting this flag the total calculation can be optimized.

Parameters
None

Return Value
FALSE(0). Calculates collisions with the first model.
TRUE(1). Calculates all collisions between models.

See Also
SetCheckAllCombinations()
IntersectionUpdateLines()

long GetCollision();

Description
Returns collision status indicating whether any collision has occurred.

Parameters
None

Return Value
Returns TRUE(1) if collision. Returns FALSE(0) if no collision.

See Also
Recalc()
IsCollision()
long GetCount();

**Description**
Returns the number of models in the model list.

**Parameters**
None

**Return Value**
Count of models

**See Also**
AddModel()  
Del Model()  
GetHeadPosition()  
GetNext()

long GetHeadPosition();

**Description**
Get head position for model list.

**Parameters**
None

**Return Value**
Returns head position of model list.

**See Also**
GetNext()  
GetCount()
long GetNext(long* pPos1);

**Description**
Gets model handle of model list and position for the next model handle.

**Parameters**
long* pPos1
A reference to a POSITION value returned by a previous GetNext, GetHeadPosition.

**Return Value**
Model handle
If the retrieved element is the last in the list, then the new value of rPosition is set to NULL.
long* pPos1
Returns a next POSITION value.

**See Also**
GetHeadPosition()
GetCount()

long GetUpdateColorForAllShapes();

**Description**
 Gets the flag status of "UpdateColorForAllShapes". The color of all shapes that are collided can be changed to the defined color, if the flag value is TRUE(1). The color of shape that is first found as collided can be changed to the defined color, if the flag value is FALSE(0).

**Parameters**
None

**Return Value**
The status of flag "UpdateColorForAllShapes".

**See Also**
SetUpdateColorForAllShapes()
UpdateColor()
long IntersectionCopyAxis6(long hModel1, long hModel2, long bIsAxis6Model);

**Description**
Copies LINE shapes from a model (hModel1) to a model (hModel2). The LINE shapes can be changed to AXIS6 shapes, if the flag of "bIsAxis6Model" is TRUE (1). The model (hModel1) should have LINE shapes if this model is the collision location.

**Parameters**
- **long hModel1**
  - hModel1 is the handle of the source model. This is the collision location.
- **long hModel2**
  - hModel2 is the handle of the destination model.
- **long bIsAxis6Model**
  - Copies LINE shapes to LINE shapes if FALSE(0). Copies LINE shapes to AXIS6 shapes if TRUE(1).

**Return Value**
The count of copied LINE shapes.

**See Also**
- IntersectionUpdateLines()
- IntersectionSetParameter()
- Recalc()
- SetCheckAll()

long IntersectionGetParameter(long nIdx1);

**Description**
Returns the status of the Intersection Parameter. The flag "bIsDeleteSamePoint" indicates whether to delete duplicate points in collision LINE shapes. The flag "bIsMakeNormalFrame" indicates whether to make the normal frame for the near polygon in collision LINE shapes. The flag "bIsSetXAxisToNextPoint" indicates whether to set the X-direction of the frame to point to the next point in the collision LINE shape.

**Parameters**
- **long nIdx1**
  - 0: Gets the status of "bIsDeleteSamePoint".
  - 1: Gets the status of "bIsMakeNormalFrame".
  - 2: Gets the status of "bIsSetXAxisToNextPoint".

**Return Value**
TRUE(1) or FALSE(0)

**See Also**
- IntersectionSetParameter()
- IntersectionCopyAxis6()
**double IntersectionGetReconnectDist();**

**Description**
Returns the distance value that is used to determine whether two points are considered identical. Lines of intersection are small segments of lines that include duplicated points. Some segments should be replaced with a single line. While the algorithm is reconnecting LINE, this distance value is used for judging two points as the same.

**Parameters**
None

**Return Value**
A distance value. The default is 0.5(mm).

**See Also**
IntersectionSetReconnectDist()
IntersectionCopyAxis6()
IntersectionUpdateLines()

**void IntersectionSetParameter(long nIdx1, long nNewValue1);**

**Description**
Sets the status of Intersection Parameter. The flag of "bIsDeleteSamePoint" sets whether to delete identical points in collision LINE shapes. The flag of "bIsMakeNormalFrame" sets whether to make the normal frame for the near polygon in collision LINE shapes. The flag of "bIsSetXAxisToNextPoint" sets whether to set the X-direction of the frame to point to the next point in the collision LINE shape.

**Parameters**
long nIdx1
0:Sets the status of "bIsDeleteSamePoint".
1: Sets the status of "bIsMakeNormalFrame".
2: Sets the status of "bIsSetXAxisToNextPoint".
long nNewValue1
TRUE(1), or FALSE(0)

**Return Value**
None

**See Also**
IntersectionGetParameter()
IntersectionSetReconnectDist()
void IntersectionSetReconnectDist(double dVal1);

**Description**
Sets the distance value that is used to determine whether two points are considered identical. Lines of intersection are small segments of lines that include duplicated points. Some segments should be replaced with a single line. While the algorithm is reconnecting LINE, this distance value is used for judging two points as the same.

**Parameters**
A distance value. The default is 0.5(mm).

**Return Value**
None

**See Also**
IntersectionGetReconnectDist()
IntersectionCopyAxis6()
IntersectionUpdateLines()

long IntersectionUpdateLines();

**Description**
Updates the Line information if collision models have LINE shapes. This function should be called after "Recalc()" function and before "IntersectionCopyAxis6()" function.

**Parameters**
None

**Return Value**
The total number of lines that collided models have.

**See Also**
IntersectionCopyAxis6()
Recalc()
long IsCollision(long hModel);

Description
Gets the collision status of the specified model.

Parameters
long hModel
Handle to the model, that is captured from "ICmdModelSvr" object.

Return Value
TRUE(1) if the model is collided. FALSE(0) if the model is not collided.

See Also
Recalc()
SetCheckAll()

void Recalc();

Description
Calculate the collision status. Use GetCollision() to check the result.

Parameters
None

Return Value
None

See Also
GetCollision()
IsCollision()

void SetCheckAll(long nNewValue);

Description
Sets the flag status of "CheckAll". This flag determines whether calculation stops after one collision is found.

Parameters
long nNewValue: TRUE (1) or FALSE(0)
CheckAll: FALSE stop the process of calculating collisions if a collision is found.
CheckAll: TRUE continue the process of calculating collisions after one is found.

Return Value
None

See Also
GetCheckAll()
IntersectionUpdateLines();
void SetCheckAllCombinations(long nNewValue1);

**Description**
Sets the status of the "nIsAllCombinations" flag. This flag controls the combinations of models checked for collision. Consider the case of three models added to "ICmdCollision" object, model A, model B, and model C. Model A is the first model to be added. If the "nIsAllCombinations" flag is set FALSE(0) the combinations of A-B and A-C are considered for collision calculation. The combinations of A-B, A-C, and B-C are considered for collision calculation, if the "nIsAllCombinations" flag is set TRUE(1). By setting this flag the total calculation can be optimized.

**Parameters**
- long nNewValue1
  - FALSE(0). Calculates collisions with the first model.
  - TRUE(1). Calculates all collisions between models.

**Return Value**
None

**See Also**
- GetCheckAllCombinations()
- Check()

void SetUpdateColorForAllShapes(long nNewValue1);

**Description**
Sets the status of the "UpdateColorForAllShapes" flag. If the flag value is set TRUE(1) the color of all the shapes that collide will be changed to the defined color. If the flag value is set FALSE(0) only the color of shape that is first found to collide will be changed to the defined color.

**Parameters**
- long nNewValue1
  - TRUE(1) Changes the color of all models that collide.
  - FALSE(0) Only changes the color of the first model found to collide.

**Return Value**
None

**See Also**
- GetUpdateColorForAllShapes()
- UpdateColor()
long UpdateColor(long nRGB);

**Description**
Changes the color used to indicate collision status.

**Parameters**
long nRGBNew color value.

**Return Value**
Number of changed color model.

**See Also**
GetCount()
SetUpdateColorForAllShapes()

class ICmdGeneralOpr : public COleDispatchDriver
double GetCycleTime(long nType1);

**Description**
Gets a Playback time and a Move time. If cell has more than one robot, the longest time is taken.

**Parameters**
long nType1
1: Move time.
2: Playback time.

**Return Value**
Time (s)

**See Also**
SetCycleTime()
double GetHeartBeat(long nType1);

**Description**
Gets Heart Beat time. If cell has more than one robot, use the "SetRobotName()" function to specify a robot before calling GetHeartBeat().

**Parameters**
long nType1
0: Robot heart beat clock
1: View heart beat clock.

**Return Value**
Heart beat clock (sec).

**See Also**
SetHeartBeat()
SetRobotName()

long GetHold();

**Description**
Gets the status of Hold. If Hold is TRUE(1), play back is held. If Hold is FALSE(0), play back is continued.

**Parameters**
None

**Return Value**
TRUE(1) or FALSE(0)

**See Also**
SetHold()

CString GetJobName();

**Description**
Gets the selected Job name.

**Parameters**
None

**Return Value**
Job name

**See Also**
SetJobName()
CString GetRobotName();

Description
Gets the selected robot name.

Parameters
None

Return Value
Robot name.

See Also
SetRobotName()

long JobReset();

Description
Resets Job sequence. This function forces execution to go to the top line of the Job.

Parameters
None

Return Value
TRUE(1), if succeed.
FALSE(0), if failed.

See Also
SetJobName()
SetRobotName()
JobStart()

long JobStart();

Description
Starts simulation with asynchronous mode.

Parameters
None

Return Value
TRUE(1), if succeed.
FALSE(0), if failed.

See Also
long JobWait(long waitTime);

**Description**
Waits until Job reaches the end of Job.

**Parameters**
- long waitTime
  - Time for wait (ms)

**Return Value**
- TRUE(1), if Job is running. Indicates the job did not complete within waitTime.
- FALSE(0), if Job has reached the end of Job.

**See Also**
- JobStart();
- JobHold();

long OnPlay(long ltype);

**Description**
Controls simulation with synchronous mode.

**Parameters**
- long ltype
  - 0: Reset Job
  - 1: Hold Job
  - 2: Play Job
  - 3: Next Job
  - 4: Pause Job

**Return Value**
- TRUE(1): Job is running.
- FALSE(0): Job reaches the end of Job.

**See Also**
- SetJobName()
void SetCycleTime(long nType1, double dNewValue1);

Description
Sets a Playback time and a Move time for reset data. If cell has two or more robots, the new value of time is set for each robot.

Parameters
long nType1
1: Move time.
2: Playback time.
double dNewValue1
The new value of time (sec)

Return Value
None

See Also
GetCycleTime()

void SetHeartBeat(long nType1, double dNewValue1);

Description
Sets Heart Beat time.

Parameters
long nType1
0: Robot heart beat clock
1: View heart beat clock.
double dNewValue1
The new value of heart beat clock (sec).

Return Value
None

See Also
GetHeartBeat()
SetRobotName()
void SetHold(long nNewValue);

**Description**
Sets the status of Hold. If Hold is TRUE(1), playback is held. If Hold is FALSE(0), playback is continued.

**Parameters**
TRUE(1) or FALSE(0)

**Return Value**
None

**See Also**
GetHold()

---

void SetJobName(LPCTSTR lpszNewValue);

**Description**
Sets the selected Job name.

**Parameters**
LPCTSTR lpszNewValue
Job name.

**Return Value**
None

**See Also**
GetJobName()

---

void SetRobotName(LPCTSTR lpszNewValue);

**Description**
Sets the selected robot name.

**Parameters**
LPCTSTR lpszNewValue
Robot name.

**Return Value**
None

**See Also**
GetRobotName()
class ICmdMatrix : public COleDispatchDriver
void GetAxis6(double* pdx, double* pdy, double* pdz, double* prx, double* pry, double* prz);

Description
Gets AXIS6 type data
Parameters
None
Return Value
double* pdx, double* pdy, double* pdz, double* prx, double* pry, double* prz
The elements of AXIS6
See Also
SetAxis6()

double GetElement(long nIdxRow, long nIdxCol);

Description
Gets the element value of a matrix.
Parameters
long nIdxRow
The index number of a row array
long nIdxCol
The index number of a column array
Return Value
The element value
See Also
SetElement()

long GetHandle();

Description
Gets the handle of a matrix. This handle is unique in an OLE connection
Parameters
None
Return Value
The handle of a matrix
See Also
Trans()
Sub()
void Inverse();

**Description**
Calculates the inverse matrix

**Parameters**
None

**Return Value**
The result is kept inside this object

**See Also**
Trans()  
Sub()  
Normalize()

void Normalize();

**Description**
Calculates the normal matrix

**Parameters**
None

**Return Value**
The result is kept inside this object

**See Also**
Inverse()

void SetAxis6(double dx, double dy, double dz, double rx, double ry, double rz);

**Description**
Sets an AXIS6 type data to the matrix

**Parameters**
double dx, double dy, double dz, double rx, double ry, double rz
The elements of an AXIS6

**Return Value**
None

**See Also**
GetAxis6()
void SetElement(long nIdxRow, long nIdxCol, double dNewValue);

**Description**
Sets the element value of a matrix

**Parameters**
- long nIdxRow
  - The index number of a row array
- long nIdxCol
  - The index number of a column array
- double dNewValue
  - The new value for the element of matrix

**Return Value**
None

**See Also**
GetElement()

void Sub(long hMat1, long hMat2, long nCalType);

**Description**
Calculates a matrix subtraction. The matrix A and the matrix B are not changed after calling this function.

**Parameters**
- long hMat1
  - The matrix handle for matrix A
- long hMat2
  - The matrix handle for matrix B
- long nCalType
  - 0: specifies calculation from a matrix A to a matrix B.
  - 1: specifies calculation from a matrix B to a matrix A.

**Return Value**
The calculated result is kept in the matrix object.

**See Also**
Trans()
void Trans(long hMatrix, long nPrePostReplace);

**Description**
Calculates a matrix translation. The matrix A is not changed after calling this function. The matrix translation is based on \(A = B \times C\) formula. A is the result, B is pre-contact matrix, C is post-contact matrix.

**Parameters**
- long hMatrix
  - The matrix handle for matrix A
- long nPrePostReplace
  - Indicates hMatrix translation.
  - 0: Pre-contact
  - 1: Post-contact
  - 2: Replace

**Return Value**
The calculated result is kept in the matrix object.

**See Also**
Sub()

**class** ICmdModel : public COleDispatchDriver
long CopyLine2Axis6();

**Description**
Copies LINE shapes to AXIS6 shapes.

**Parameters**
None

**Return Value**
The number of copied shape.

**See Also**
DeleteAllLines()
LineReConnect()
long DeleteAllLines();

**Description**
Deletes all LINE shapes in this model.

**Parameters**
None

**Return Value**
The number of deleted shapes

**See Also**
DeleteShape()

long DeleteAllShapes(LPCTSTR strType);

**Description**
Deletes all shapes that are the specified shape type. Valid shape types are AXIS6, LINE, etc.

**Parameters**
LPCTSTR strType
The shape type

**Return Value**
The number of deleted shapes.

**See Also**
DeleteAllLines()
DeleteShapes()

long DeleteShape(long hShape);

**Description**
Deletes specified shape.

**Parameters**
long hShape
The handle of shape

**Return Value**
TRUE(1), if succeed.
FALSE(0), if failed.

**See Also**
ShapeGetHeadPosition()
ShapeGetNext()
double GetAttributeDoubleByIndex(long nIdx1);

Description
Gets the attribute status by the index.

Parameters
long nIdx1
0: gets the opacity value.

Return Value
Returns the "double" value.

See Also
SetAttributeDoubleByIndex()

long GetAttributeLongByIndex(long nIdx1);

Description
Gets the attribute status by the index.

Parameters
long nIdx1
0: Dummy. Indicates the dummy model.
1: HideSee. Indicates the HideSee value.
2: Color. Indicates the color of model.
3: IsDirty. Indicates the flag of IsDirty.
4: Dirty Count. Indicates the number of the dirty count.
5: Is MDL model. Indicates that the model has MDL file.
6: Has IO object. Indicates that the model has IO object.
7: mIsRobot. Indicates that the model is a part of robot arm.

Return Value
Returns the "long" value.

See Also
SetAttributeLongByIndex()
CString GetFileName();

**Description**
Gets the file name path of model. This can be *.mdl file or *.rwx file.

**Parameters**
None.

**Return Value**
The file name path.

**See Also**
SetFileName()

long GetHandle();

**Description**
Gets handle of the model.

**Parameters**
None.

**Return Value**
The handle of model.

**See Also**

long GetHideSee();

**Description**
Gets the flag of Hide/See.

**Parameters**
None

**Return Value**
TRUE(1), if the model is visible.
FALSE(0), if the model is invisible.

**See Also**
SetHideSee()
long GetMatrix(long hRefModel, long hMatrix);

**Description**
Gets the model location to the matrix object.

**Parameters**
- long hRefModel
  Indicates the reference model for location. If NULL(0), the location is the same location as the local location.
- long hMatrix
  The handle of matrix to be stored the location data.

**Return Value**
- TRUE(1), if succeed.
- FALSE(0), if failed.

**See Also**
- GetPosition()
- SetMatrix()

CString GetModelType();

**Description**
Gets the model type that user defined. The user application can use this name.

**Parameters**
- None

**Return Value**
The model type strings.

**See Also**
- SetModelType()

CString GetName();

**Description**
Gets the model name.

**Parameters**
- None

**Return Value**
The name of model.

**See Also**
- SetName()
4.32 OLE

long GetParent();

**Description**
Gets the handle of the parent model.

**Parameters**
None

**Return Value**
The handle of the parent model.

**See Also**
SetParent()
MoveParent()

long GetPosition(long hRefModel, double* pdPx, double* pdPy, double* pdPz,
                double* pdRx, double* pdRy, double* pdRz);

**Description**
Gets AXIS6 data of the model location.

**Parameters**
long hRefModel
Indicates the reference model for location. If NULL(0), the location is the same location as the local location.

**Return Value**
double* pdPx, double* pdPy, double* pdPz, double* pdRx, double* pdRy, double* pdRz
The elements of AXIS6.
TRUE(1), if succeed.
FALSE(0), if failed.

**See Also**
SetPosition()
GetMatrix()
long GetShapeColor(long hShape, long nToFile);

**Description**

**Parameters**
- long hShape
  - The handle of the shape.
- long nToFile
  - Indicated the file space color, if TRUE(1).
  - Indicated the memory space color, if FALSE(0).

**Return Value**
- TRUE(1), if succeed.
- FALSE(0), if failed.

**See Also**
- SetShapeColor()

double GetShapeData(long hShape, long nRow, long nCol);

**Description**
GETS A SHAPE DATA.

**Parameters**
- long hShape
  - The shape handle
- long nRow
  - The row number of shape array
- long nCol
  - The column number of shape array

**Return Value**
- The element data.

**See Also**
- SetShapeData()
- GetShapeData6()
long GetShapeData6(long hShape, long nRow, double* lpdVal1, double* lpdVal2, double* lpdVal3, double* lpdVal4, double* lpdVal5, double* lpdVal6);

**Description**
Gets shape data as AXIS6 type. This call function can be faster than calling "GetShapeData()" function.

**Parameters**
- long hShape
  - The shape handle
- long nRow
  - The row number of shape array

**Return Value**
- TRUE(1), if succeed
- FALSE(0), if failed

**See Also**
- SetShapeData6()
- GetShapeData()

CString GetShapeName(long hShape);

**Description**
Gets the Shape Name that a user assigns to a shape. The Shape Name is a unique attribute.

**Parameters**
- long hShape
  - The handle of shape

**Return Value**
- The name string

**See Also**
- SetShapeName()
double LineGetReconnectDistance();

**Description**
Gets the maximum distance between two points that allows them to be merged.

**Parameters**
None

**Return Value**
The distance value

**See Also**
LineSetReconnectDistance()

long LineReConnect(long bIsDeleteSamePoint);

**Description**
Merges a LINE2 shape to a LINE shape.

**Parameters**
long bIsDeleteSamePoint
The same points are deleted, if TRUE(1)
The same points are not deleted, if FALSE(0)

**Return Value**
The number of shapes that have been merged

**See Also**
LineSetReconnectDistance()

void LineSetReconnectDistance(double dValue1);

**Description**
Sets the maximum distance between two points that allows them to be merged.

**Parameters**
double dValue1
The new value of distance

**Return Value**
None

**See Also**
LineGetReconnectDistance()
long LoadFile();

**Description**
Loads file shapes to the memory for viewing.

**Parameters**
None

**Return Value**
TRUE(1), if succeed.
FALSE(0), if failed.

**See Also**
SaveFile()
RefreshView()

long MovParent(long nNewValue);

**Description**
Sets the handle of the parent model. If the new chosen parent is not at the same position as the old parent then the model will move relative to the world frame. The model will move so that its location relative to the new parent is the same as its previous location relative to the old parent.

**Parameters**
long nNewValue
The handle of the new parent model

**Return Value**
None

**See Also**
SetParent()

long NewShape(LPCTSTR strType, long color, long rowmax);

**Description**
Creates a new shape.

**Parameters**
LPCTSTR strType
The shape type: AXIS6, LINE, BOX, etc...
long rowmax
The number of rows that the shape has

**Return Value**
The handle of the new shape

**See Also**
DeleteShape()
long RefreshView();

**Description**
Refreshes the model view. Call this function if shapes were changed.

**Parameters**
None

**Return Value**
TRUE(1), if succeed
FALSE(0), if failed

**See Also**
SetShapeColor()

long ReLocateAllShapes(long nhModel2);

**Description**
Relocate shapes based on the specified model. This allows recalculation of the local location of shapes, for example, changing TOOL-tcp.

**Parameters**
long nhModel2
Relocation model frame.

**Return Value**
TRUE(1), if succeed.
FALSE(0), if failed.

**See Also**

long SaveFile();

**Description**
Saves shapes to the file for storing. This function works with *.mdl file. This function does not work with *.rwx file.

**Parameters**
None

**Return Value**
TRUE(1), if succeed.
FALSE(0), if failed.

**See Also**
LoadFile()
RefreshView()
long SetAttributeDoubleByIndex(long nIdx1, double dNewValue1);

**Description**
Sets the attribute status by the index.

**Parameters**
- long nIdx1
- 0: the opacity value.
- double dNewValue1
  - The new value.

**Return Value**
- TRUE(1), if succeed.
- FALSE(0), if failed.

**See Also**
GetAttributeDoubleByIndex()

long SetAttributeLongByIndex(long nIdx1, long nNewValue1);

**Description**
Sets the attribute status specified by the index.

**Parameters**
- long nIdx1
- 0: Dummy. Indicates the dummy model.
- 1: HideSee. Indicates the HideSee value.
- 2: Color. Indicates the color of model.
- 3: IsDirty. Indicates the flag of IsDirty.
- 4: Dirty Count. Indicates the number of the dirty count.
- 5: Is MDL model. Indicates that the model has MDL file.
- 6: Has IO object. Indicates that the model has IO object.
- 7: mIsRobot. Indicates that the model is a part of robot arm.
- long nNewValue1
  - The new value.

**Return Value**
- TRUE(1), if succeed.
- FALSE(0), if failed.

**See Also**
GetAttributeLongByIndex()
void SetFileName(LPCTSTR lpszNewValue);

**Description**
Sets the file name path of the model. This can be a *.mdl file or a *.rwx file.

**Parameters**
LPCTSTR lpszNewValue.
The file name path.

**Return Value**
None.

**See Also**
GetFileName()

void SetHandle(long nNewValue);

**Description**
Sets the model handle. The model handle is a unique value in the cell.

**Parameters**
long nNewValue
The model handle

**Return Value**
None

**See Also**
GetHandle()

void SetHideSee(long nNewValue);

**Description**
Sets the flag of Hide/See.

**Parameters**
long nNewValue
See the model, if TRUE(1).
Hide the model, if FALSE(0).

**Return Value**
None

**See Also**
GetHideSee()
void SetMatrix(long hRefModel, long hMatrix, long nNewValue);

**Description**
Sets the model location as specified by the matrix object.

**Parameters**
long hRefModel
Indicates the reference model relative to which the location is set. If NULL(0), the new location is set relative to the local location.
long hMatrix
The handle of the matrix in which the new location data is stored.
long nNewValue
if TRUE(1) Calculate the new locations of the child models immediately
if FALSE(0) Delay child calculations until a Refresh function is called

**Return Value**
None

**See Also**
GetPosition()
GetMatrix()

void SetModelType(LPCTSTR lpszNewValue);

**Description**
Sets the model type that user defined. The user application can use this name.

**Parameters**
LPCTSTR lpszNewValue
The new model type that the user has defined.

**Return Value**
None

**See Also**
GetModelType()
void SetName(LPCTSTR lpszNewValue);

**Description**
Sets the model name.

**Parameters**
LPCTSTR lpszNewValue
The new name of the model.

**Return Value**
None

**See Also**
GetName()

void SetParent(long nNewValue);

**Description**
Sets the handle of the parent model. The model location relative to the world frame is not changed. The local location data to relative to the new parent is changed.

**Parameters**
long nNewValue
The handle of the new parent model

**Return Value**
None

**See Also**
GetParent()
MovParent()
void SetPosition(long hRefModel, double dPx, double dPy, double dPz, double dRx, double dRy, double dRz, long nNewValue);

Description
Sets AXIS6 data of the model location.

Parameters
long hRefModel
Indicates the reference model for the new location. If NULL(0), the reference location is the same location as the local location.
double dPx, double dPy, double dPz, double dRx, double dRy, double dRz
The elements of AXIS6 data.
long nNewValue
Calculate the location of child models immediately, if TRUE(1). Delay child calculations until a Refresh function is called, if FALSE(0).

Return Value
None

See Also
GetPosition()
SetMatrix()

void SetShapeColor(long hShape, long nToFile, long nNewValue);

Description
Sets the shape color. Each shape in the model has a separate color attribute in the memory space and the file space. The color you see is the color of the memory space. The color the model file keeps is the color of the file space. In most cases both colors are the same, but they could be different. For example the collision function could change a model's memory space color to RED.

Parameters
long hShape
The handle of the shape.
long nToFile
Set the file space color, if TRUE(1).
Set the memory space color, if FALSE(0).
long nNewValue
The new value

Return Value
None

See Also
GetShapeColor()
```c
void SetShapeData(long hShape, long nRow, long nCol, double dNewValue);

  **Description**
  Sets a shape data.

  **Parameters**
  long hShape
  The shape handle
  long nRow
  The row number of shape array
  long nCol
  The column number of shape array
  double dNewValue
  The new value

  **Return Value**
  None

  **See Also**
  GetShapeData()

long SetShapeData6(long hShape, long nRow, double dVal1, double dVal2, double
dVal3, double dVal4, double dVal5, double dVal6);

  **Description**
  Sets a shape data as AXIS6 type. Calling this function can be faster than calling the "SetShapeData()"
  function.

  **Parameters**
  long hShape
  The shape handle
  long nRow
  The row number of shape array
  double dVal1, double dVal2, double dVal3, double dVal4, double dVal5, double dVal6
  The column data element

  **Return Value**
  TRUE(1), if succeed
  FALSE(0), if failed

  **See Also**
  GetShapeData6()
  SetShapeData()
```
void SetShapeName(long hShape, LPCTSTR lpszNewValue);

**Description**
Sets the shape name as specified. A shape name is a unique attribute.

**Parameters**
- **long hShape**
  The handle of shape
- **LPCTSTR lpszNewValue**
  The new name string

**Return Value**
None

**See Also**
GetShapeName()

long ShapeCopyRow(long nhShape1, long nIdx1Start, long nhModel2, long nhShape2, long nIdx2Start, long nCount1);

**Description**
Copy shape data from the source shape to the destination shape. The type of shape should be the same.
This function can be used for LINE, AXIS6 shape types.

**Parameters**
- **long nhShape1**
  The shape handle for destination
- **long nIdx1Start**
  The index number for the start of writing
- **long nhModel2**
  The model handle for source
- **long nhShape2**
  The shape handle for source
- **long nIdx2Start**
  The index number for the start of reading
- **long nCount1**
  The total number of elements to be copied

**Return Value**
TRUE(1), if succeed.
FALSE(0), if failed.

**See Also**
ShapeGetRowColMax()
long ShapeGetCount();

Description
Gets the total number of shapes in the model.
Parameters
None
Return Value
The total number of shapes
See Also
ShapeGetRowColMax()
NewShape()

long ShapeGetHeadPosition();

Description
Gets the head position handle of the shape list.
Parameters
None
Return Value
The head position
See Also
ShapeGetNext()
ShapeGetCount()

long ShapeGetNext(long* pPos);

Description
Gets the shape handle.
Parameters
long* pPos
The handle for a shape
Return Value
The shape handle
See Also
ShapeGetHeadPosition()
ShapeGetCount()
long ShapeGetRowColMax(long hShape, long* plRow, long* plCol);

**Description**
Gets the total number of rows and columns that the shape has.

**Parameters**
- long hShape: The shape handle

**Return Value**
- long* plRow: The total number of rows that the shape has
- long* plCol: The total number of columns that the shape has

**See Also**
- ShapeGetCount()
long ShapeMerge(long nhShape1, long nhShape2);

Description
Merges shapes. Both shapes should be the same shape type. This function can be used for shapes like AXIS6, LINE etc.

Parameters
long nhShape1
The first shape handle
long nhShape2
The second shape handle

Return Value
The shape handle that is created from shape1 and shape2

See Also
ShapeCopyRow()

long ShapeRefreshView(long hShape);

Description
Refreshes the shape view. Call this function, if you changed shape color or data.

Parameters
long hShape
The shape handle

Return Value
TRUE(1), if succeed
FALSE(0), if failed

See Also
class ICmdModelSvr : public COleDispatchDriver

long ConvertMatrix(long hModel1, long hMat1, long hModel2, long hMat2);

**Description**
The output is Matrix 2. It is at the same position as Matrix 1 and it specifies this position relative to Model 2.

**Parameters**
- long hModel1
  - The model handle for hMat1
- long hMat1
  - The matrix handle for source
- long hModel2
  - The model handle for hMat2
- long hMat2
  - The matrix handle for destination

**Return Value**
TRUE(1), if succeed
FALSE(0), if failed

**See Also**
GetHandle()

long Delete(long hModel);

**Description**
Deletes model

**Parameters**
- long hModel
  - The model handle

**Return Value**
TRUE(1), if succeed
FALSE(0), if failed

**See Also**
New()
GetHandle()
long GetChildItem(long hModel1);

Description
Gets the model handle of the child model

Parameters
long hModel1
The model handle

Return Value
The model handle

See Also
GetNextItem()

long GetHandle(LPCTSTR strName);

Description
Get the model handle from string

Parameters
LPCTSTR strName
The model name string

Return Value
The model handle

See Also
New()

CString GetModelPath();

Description
Gets the model path of the cell.
This path is produced by using "Cell Path" + "Models".

Parameters
None

Return Value
The model path name string

See Also
NewCmdModel()
CString GetName(long hModel);

**Description**
Gets the model name string

**Parameters**
long hModel
The model handle

**Return Value**
The model name string

**See Also**
GetHandle()
SetName()

long GetNextItem(long hModel1, long nCode);

**Description**
Gets the model handle in the Cad tree.

**Parameters**
long hModel1
The model handle
long nCode
TVGN_ROOT 0x0000 Indicates the root model.
TVGN_NEXT 0x0001 Indicates the next model.
TVGN_PREVIOUS 0x0002 Indicates the previous model.
TVGN_PARENT 0x0003 Indicates the parent model.
TVGN_CHILD 0x0004 Indicates the child model.

**Return Value**
The model handle

**See Also**
GetChildItem()
CString GetType(long hModel1);

**Description**
Gets the type name string that the user made.

**Parameters**
- long hModel1
  - The model handle

**Return Value**
The type name string

**See Also**
- SetType()

long New(LPCTSTR strName);

**Description**
Creates a new model.

**Parameters**
- LPCTSTR strName
  - The name string of model

**Return Value**
- TRUE(1), if succeed
- FALSE(0), if failed

**See Also**
- Delete()

LPDISPATCH NewCmdCollision(long hModel1, long hModel2);

**Description**
Creates the CmdCollision object

**Parameters**
- long hModel1
  - The first model handle
- long hModel2
  - The second model handle

**Return Value**
The dispatch handle

**See Also**
- ICmdCollision object
LPDISPATCH NewCmdMatrix();

Description
Creates the CmdMatrix object

Parameters
None

Return Value
The dispatch handle

See Also
ICmdMatrix object

LPDISPATCH NewCmdMIOBitConnect();

Description
Creates the CmdMIOBitConnect object

Parameters
None

Return Value
The dispatch handle

See Also
ICmdCollision object

LPDISPATCH NewCmdModel(long hModel);

Description
Creates the CmdModel object

Parameters
long hModel
The model handle

Return Value
The dispatch handle

See Also
ICmdModel object
LPDISPATCH NewCmdModelIO(long hModel);

Description
Creates the CmdModelIO object

Parameters
long hModel
The model handle

Return Value
The dispatch handle

See Also
ICmdModelIO object

LPDISPATCH NewCmdModelType(LPCTSTR strType1);

Description
Creates the CmdModelType object. This object creates a list of models that have a specified model type.

Parameters
LPCTSTR strType1
Model type

Return Value
The dispatch handle

See Also
ICmdModelType object

LPDISPATCH NewCmdProcessAngle();

Description
Creates the CmdProcessAngle object

Parameters
None

Return Value
The dispatch handle

See Also
ICmdProcessAngle object
void Recalc();

**Description**
Recalculates the location of model. Call this function when the location of a model has been changed.

**Parameters**
None

**Return Value**
None

**See Also**

long RelocateModel(long nTag1, LPCTSTR strTypeName);

**Description**
The first argument defines the operation type. The value "1" is the only operation in the current implementation. This operation works with AXIS6 Shape name. AXIS6 Shape has two elements, f1 and f2. Each of these is an AXIS6 value. The second argument is a Shape Type Name. This may be a user-defined type, such as "RELOCATE". This function will operate on all models that have the type name specified by strTypeName, "RELOCATE". The positions of these models relative to their parents will be changed according to the AXIS6 values in f1. The positions of the children of these models will be changed according to the AXIS6 values in f2. It may be helpful to preserve the original locations in [save].mdl.

**Parameters**
long nTag1
1: The AXIS6 values for the relocate operations
LPCTSTR strTypeName
The shape type name that specifies models to be relocated

**Return Value**
TRUE(1), if succeed
FALSE(0), if failed

**See Also**
void SetName(long hModel, LPCTSTR lpszNewValue);

Description
Sets the model name string. This function renames the model.

Parameters
long hModel
The model handle
LPCTSTR lpszNewValue
The new model name string

Return Value
None

See Also
GetHandle()
GetName()

void SetType(long hModel1, LPCTSTR lpszNewValue);

Description
Sets the model type. This type name can be used by any application for categorizing models.

Parameters
long hModel1
The model handle
LPCTSTR lpszNewValue
The new type name string

Return Value
None

See Also
GetType()
class ICmdMotoPosition : public COleDispatchDriver
long AttrGetHandle(long nTag1);

Description
Gets the handle of attribute slots. The attribute slot has two categories. One is the robot configuration handle; other is the reference frame handle. ICmdMotoRobot object can provide the handle information.

Parameters
long nTag1
0: Indicates the robot configuration. (RCONF)
1: Indicates the reference frame. (RefFrame)

Return Value
The handle of an attribute.

See Also
ICmdMotoRobot
AttrSetHandle()

long AttrSetHandle(long nTag1, long nhAttr1);

Description
This function sets the handle to either a Robot Configuration handle (RCONF) or a Reference Frame handle (RefFrame). The nhAttr is supplied by AttrGetHandle().

Parameters
long nTag1
0: Indicates the robot configuration. (RCONF)
1: Indicates the reference frame. (RefFrame)
long nhAttr1
The handle of attribute slots

Return Value
TRUE(1), if succeed
FALSE(0), if failed

See Also
ICmdMotoRobot
AttrGetHandle()
long JointGetDefaultAsPositionType(long nTag1);

**Description**
Initializes the type of a position object and sets the values to zero.

**Parameters**
long nTag1
0: Indicates Pulse data.
1: Indicates Rectangular data.
2: Indicates Angle data.

**Return Value**
TRUE(1), if succeed
FALSE(0), if failed

**See Also**
JointGetXyzRxyz();
JointGetValueDouble();
JointGetValueLong();

double JointGetValueDouble(long nIdx1);

**Description**
Gets the value of a double type. The value can be Angle data.

**Parameters**
long nIdx1
The index number of the Double array value to be returned, 0 - 7.

**Return Value**
The value of the Double array element

**See Also**
JointSetValueDouble();

long JointGetValueLong(long nIdx1);

**Description**
Gets the value of a long type. The value can be Pulse data.

**Parameters**
long nIdx1
The index number of the Long array to be returned, 0 - 7

**Return Value**
The value of the Long array

**See Also**
JointSetValueLong();
long JointGetXyzRxyz(double* pdVal1, double* pdVal2, double* pdVal3, double* pdVal4, double* pdVal5, double* pdVal6);

**Description**
Gets AXIS6 (X,Y,Z,Rx,Ry,Rz) data.

**Parameters**
double* pdVal1, double* pdVal2, double* pdVal3, double* pdVal4, double* pdVal5, double* pdVal6

The elements of an AXIS6 data

**Return Value**
TRUE(1), if succeed
FALSE(0), if failed

**See Also**
JointGetXyzRxyz()

long JointSetValueDouble(long nIdx1, double dNewValue1);

**Description**
Sets the value of a double type. The value can be Angle data.

**Parameters**
long nIdx1
The index number of the Double array element to be set
double dNewValue1
The new value to be set in the Double array

**Return Value**
TRUE(1), if succeed
FALSE(0), if failed

**See Also**
JointGetValueDouble()
long JointSetValueLong(long nIdx1, long nNewValue1);

**Description**
Sets the value of a long type. The value can be Pulse data.

**Parameters**
long nIdx1
The index number of the Long array to be set
double dNewValue1
The value to be set in the Long array

**Return Value**
TRUE(1), if succeed
FALSE(0), if failed

**See Also**
JointGetValuelong()

long JointSetXyzRxyz(double dVal1, double dVal2, double dVal3, double dVal4, double dVal5, double dVal6);

**Description**
Sets AXIS6 (X,Y,Z,Rx,Ry,Rz) data.

**Parameters**
double dVal1, double dVal2, double dVal3, double dVal4, double dVal5, double dVal6
The elements of an AXIS6 data

**Return Value**
TRUE(1), if succeed
FALSE(0), if failed

**See Also**
JointGetXyzRxyz()
long PrpGetPositionType();

**Description**
Gets the position type. The position type is one of "Pulse", "XyzRxyz", or "Angle" type.

**Parameters**
None

**Return Value**
0: indicates "PULSE" type.
1: indicates "XyzRxyz" type.
2: indicates "Angle" type.

**See Also**
PrpSetPositionType()

long PrpSetPositionType(long nNewValue);

**Description**
Sets the position type. The position type is one of "Pulse", "XyzRxyz", or "Angle" type.

**Parameters**
long nNewValue
0: indicates "PULSE" type.
1: indicates "XyzRxyz" type.
2: indicates "Angle" type.

**Return Value**
Returns the same value of nNewValue.

**See Also**
PrpGetPositionType()
class ICmdMotoRobot : public COleDispatchDriver
long AttrGetCount(long nTag1);

Description
Gets the count of attribute handles for the robot configuration or the reference frame.

Parameters
long nTag1
0: Indicates the robot configuration. (RCONF)
1: Indicates the reference frame. (RefFrame)

Return Value
The handle for RCONF or RefFrame

See Also
AttrGetHeadPosition()
AttrGetItemText()
AttrGetNext()

long AttrGetDefaultRect(LPDISPATCH lpdispatch);

Description
Sets the default attribute of an ICmdMotoPosition object to rectangular.

Parameters
LPDISPATCH lpdispatch
The object of ICmdMotoPosition

Return Value
TRUE(1), if succeed
FALSE(0), if failed

See Also
AttrSetDefaultRect()
long AttrGetHeadPosition(long nTag1);

**Description**
 Gets the head position handle for the position attributes.

**Parameters**
- long nTag1
  - 0: Indicates the robot configuration. (RCONF)
  - 1: Indicates the reference frame. (RefFrame)

**Return Value**
The handle of the list for the attributes

**See Also**
- AttrGetNext()
- AttrGetCount()
long AttrGetNext(long nTag1, long nPos1, long* pnPos2);

Description
Gets a handle of the rectangular attribute.

Parameters
long nTag1
0: Indicates the robot configuration. (RCONF)
1: Indicates the reference frame. (RefFrame)
long nPos1
The position handle of the list for the input
long* pnPos2
The position handle of the list for the output

Return Value
TRUE(1), if succeed
FALSE(0), if failed

See Also
AttrGetCount()
AttrGetHeadPosition()

long AttrSetDefaultRect(LPDISPATCH lpdispatch);

Description
Sets the default attribute as same as the ICmdMotoPosition object. The content of this function is not implemented.

Parameters
LPDISPATCH lpdispatch
The object of ICmdMotoPosition

Return Value
TRUE(1), if succeed
FALSE(0), if failed

See Also
AttrGetDefaultRect()
long GetCurrentJobLine();

Description
Gets the current Job-line number.

Parameters
None

Return Value
The number of the Job-line.

See Also
SetCurrentJobLine()
SetCurrentJobName()

CString GetCurrentJobName();

Description
Gets the current Job-name.

Parameters
None

Return Value
The current Job-name.

See Also
SetCurrentJobName

long GetCurrentJobStep();

Description
Gets the current Job-step number. The "Job-step" is the "Job-line" which has the current robot position.

Parameters
None

Return Value
The number of the Job-step.

See Also
SetCurrentJobStep()
SetCurrentJobName()
long GetCurrentToolHandle();

Description
Gets the current Tool-handle.

Parameters
None

Return Value
The Tool-handle

See Also
SetCurrentToolHandle()

double GetCycleTime(long nType1);

Description
Gets the value of "Move Time" or "Play Time". The "Move Time" is the time that a robot was moving during the job. The "Play Time" is the time that was required for the job to be complete; this includes the "Move Time" and "Timer" time etc.

Parameters
long nType1
1: Selects "Move Time"
2: Selects "Play Time"

Return Value
The time (sec)

See Also
SetCycleTime()

long GetHandle();

Description
Gets the robot handle.

Parameters
None

Return Value
The handle of the robot

See Also
CString GetMdlFilePath();

**Description**
Gets the string of the robot model file.

**Parameters**
None

**Return Value**
The robot model file

**See Also**
SetMdlFilePath()

long GetMotionModuleCode();

**Description**
Gets the code of the Motion module. The "RRS module" will supply an accurate cycle time. The "Internal module" will supply a quick simulation.

**Parameters**
None

**Return Value**
TRUE(1), indicates the robot uses RRS module for the path planning.
FALSE(0), indicates the robot uses the Internal module for the path planning.

**See Also**
SetMotionModuleCode()

long GetPowerStatus(long nType1);

**Description**
Gets the power status of the robot.

**Parameters**
long nType1
1: indicates the power supply of the robot
other: not implemented

**Return Value**
TRUE(1), if the power is on.
FALSE(0), if the power is off.

**See Also**
SetPowerStatus()
CString GetRobotPath();

_description_
Gets the robot path of the folder that holds the robot information.

(parameters)
None

_return_value_
The path name of the robot.

_see also_
SetRobotPath()

long JobGetHeadPosition();

_description_
Gets the position handle of the Job list.

(parameters)
None

_return_value_
The head position handle of the Job list.

_see also_
JobGetNext()

CString JobGetNext(long* pPos);

_description_
Gets the Job name in the job list.

(parameters)
long* pPos
The desired position number in the Job list

_return_value_
the Job name string

_see also_
JobGetHeadPosition()
long MpsConvertPosition(long nhRbModule, LPDISPATCH lpdispatch1, LPDISPATCH lpdispatch2);

**Description**
Converts a robot position object between the "Pulse" and the "Rectangular". The type of position objects should be defined (as "Pulse" or "Rectangular") before calling this function.

**Parameters**
- long nhRbModule
  Indicates the module of Robot, for instance the robot module or the external axis module.
- LPDISPATCH lpdispatch1
  The source object of a robot position (ICmdMotoPosition)
- LPDISPATCH lpdispatch2
  The destination object of a robot position (ICmdMotoPosition)

**Return Value**
- TRUE(1), if succeed
- FALSE(0), if failed

**See Also**
- MpsGetPositionCur()

long MpsGetPositionCur(long nhRbModule, LPDISPATCH lpdisp1);

**Description**
Gets a current position of the robot. The position type should be defined before calling.

**Parameters**
- long nhRbModule
  Indicates the module of Robot, for instance the robot module or the external axis module.
- LPDISPATCH lpdisp1
  The source object of a robot position (ICmdMotoPosition)

**Return Value**
- TRUE(1), if succeed
- FALSE(0), if failed

**See Also**
- MpsConvertPosition()
  - MpsSetPositionCur()
long MpsSetPositionCur(long nhRbModule, LPDISPATCH lpdispatch);

**Description**
Sets a current position of the robot. The position type should be defined before calling.

**Parameters**
- long nhRbModule
  Indicates the module of Robot, for instance the robot module or the external axis module.
- LPDISPATCH lpdispatch1
  The source object of a robot position (ICmdMotoPosition)

**Return Value**
- TRUE(1), if succeed
- FALSE(0), if failed

**See Also**
MpsConvertPosition()
MpsGetPositionCur()

long MpsSetPositionCurTcpSymmetry(long nhRbModule, LPDISPATCH lpdispatch);

**Description**
Sets a current position of the robot using the symmetry logic. The position type should be the "Rectangular" type. The symmetry logic helps to find the minimum motion for the new position from the current robot position.

**Parameters**
- long nhRbModule
  Indicates the module of Robot, for instance the robot module or the external axis module.
- LPDISPATCH lpdispatch1
  The source object of a robot position (ICmdMotoPosition)

**Return Value**
- TRUE(1), if succeed
- FALSE(0), if failed

**See Also**
MpsConvertPosition()
MpsGetPositionCur()}
LPDISPATCH NewCmdMotoJob(LPCTSTR strJobName);

**Description**
Creates the ICmdMotoJob object.

**Parameters**
LPCTSTR strJobName
The job name

**Return Value**
The dispatch handle

**See Also**

LPDISPATCH NewCmdMotoRobotMemory();

**Description**
Creates the ICmdMotoRobotMemory object.

**Parameters**
None

**Return Value**
The dispatch handle

**See Also**

LPDISPATCH NewCmdMotoRobotSystem();

**Description**
Creates the ICmdMotoRobotSystem object.

**Parameters**
None

**Return Value**
The dispatch handle

**See Also**
long RbModuleGetCount();

**Description**
Gets the count of the number of motion modules. The motion module may be the RRS module or the internal motion module.

**Parameters**
None

**Return Value**
The number of motion modules (usually two)

**See Also**
RbModuleGetHeadPosition()
RbModuleGetNext()

long RbModuleGetHeadPosition();

**Description**
Gets the head position of the motion module list.

**Parameters**
None

**Return Value**
The head position of the motion module list

**See Also**
RbModuleGetNext()

CString RbModuleGetItemText(long nhRbModule);

**Description**
Gets the motion module name.

**Parameters**
long nhRbModule
The motion module handle

**Return Value**
The motion module name
"r1", "s1", etc...

**See Also**
RbModuleGetHeadPosition()
RbModuleGetNext()
long RbModuleGetJointCount(long nhRbModule);

**Description**
Gets the joint count of the robot module.

**Parameters**
long nhRbModule
The module handle

**Return Value**
The joint count

**See Also**
RbModuleGetItemText()

long RbModuleGetNext(long* pnPos1);

**Description**
Gets the module handle of the next robot.

**Parameters**
long* pnPos1
The position of the module list

**Return Value**
The module handle of the robot

**See Also**
RbModuleGetHeadPosition()

long RbModuleJointGetAxisBits(long nhRbModule);

**Description**
Gets the axis bits of a robot module.

**Parameters**
long nhRbModule
The module handle of the robot

**Return Value**
The axis bits of a robot module

**See Also**
RbModuleGetJointCount()
void SetCurrentJobLine(long nNewValue);

**Description**
Sets the current Job-line number.

**Parameters**
- long nNewValue
  - The number of the Job-line

**Return Value**
None

**See Also**
GetCurrentJobLine()
GetCurrentJobName()

void SetCurrentJobName(LPCTSTR lpszNewValue);

**Description**
Sets the current Job-name.

**Parameters**
- LPCTSTR lpszNewValue
  - The current Job-name

**Return Value**
None

**See Also**
GetCurrentJobName

void SetCurrentToolHandle(long nTool1);

**Description**
Sets the current Tool-handle.

**Parameters**
- long nTool1
  - The Tool-handle

**Return Value**
None

**See Also**
GetCurrentToolHandle()
void SetCycleTime(long nType1, double dNewValue1);

Description
Sets the value of "Move Time" or "Play Time". The "Move Time" is the time that a robot was moving during
the job. The "Play Time" is the time that was required for the job to be complete; this includes the "Move
Time" and "Timer" time etc. This function can be used to reset the time data.

Parameters
long nType1
1: indicates "Move Time"
2: indicates "Play Time"
double dNewValue1
The new time for initializing

Return Value
None

See Also
GetCycleTime()

void SetMdlFilePath(LPCTSTR lpszNewValue);

Description
Sets the string of the robot model file.

Parameters
LPCTSTR lpszNewValue
The robot model file

Return Value
None

See Also
GetMdlFilePath()
void SetMotionModuleCode(long nNewValue);

**Description**
Sets the code of Motion module. The "RRS module" will supply an accurate cycle time. The "Internal module" will supply a quick simulation.

**Parameters**
- long nNewValue
  - TRUE(1), indicates the robot uses RRS module for the path planning.
  - FALSE(0), indicates the robot uses the Internal module for the path planning.

**Return Value**
None

**See Also**
GetMotionModuleCode()

void SetPowerStatus(long nType1, long nNewValue1);

**Description**
Sets the power status of the robot.

**Parameters**
- long nType1
  - 1: indicates the power supply of the robot
  - other: not implemented
- long nNewValue1
  - TRUE(1), sets the power on.
  - FALSE(0), sets the power off.

**Return Value**
None

**See Also**
GetPowerStatus()
class ICmdProcessAngle : public COleDispatchDriver

long AdjustAfterMulMatrix(long nPos1, long nIdx1, double dNewValue1);

Description
Adjusts a frame with AXIS6 data. The AXIS6 type frame is constructed with X, Y, Z, Rx, Ry, Rz elements. Only one element can be changed at a time.

Parameters
long nPos1
The position handle of a frame
long nIdx1
Selects an element of AXIS6 (X, Y, Z, Rx, Ry, Rz), starts with 0.
double dNewValue1
The new value must be millimeters for X, Y, Z or radians for Rx, Ry, Rz.

Return Value
TRUE(1), if succeed
FALSE(0), if failed

See Also
AdjustAfterMulMatrixAll()

long AdjustAfterMulMatrixAll(long nIdx1, double dNewValue1);

Description
Adjusts frames with AXIS6 data. The AXIS6 type frame is constructed with X, Y, Z, Rx, Ry, Rz elements. Only one element can be changed in all the frames of the model.

Parameters
long nIdx1
Selects an element of AXIS6 (X, Y, Z, Rx, Ry, Rz), starts with 0.
double dNewValue1
The new value must be millimeters for X, Y, Z or radians for Rx, Ry, Rz units for (Rx, Ry, Rz).

Return Value
TRUE(1), if succeed
FALSE(0), if failed

See Also
AdjustAfterMulMatrix ()
long AdjustGetCondition(long nIdx1);

**Description**
Gets the setting of an adjust parameter. Currently the only supported adjustment is adjusting the Z-direction. When this parameter is set to TRUE the Z-direction of frames will be adjusted to have the same direction as the first frame.

**Parameters**
long nIdx1
Indicates the type of adjusting. The only supported value is 0.

**Return Value**
TRUE(1), if a parameter is On.
FALSE(0), if a parameter is Off.

**See Also**
AdjustSetCondition()

long AdjustPathAllAsCylindricalFrame(long hModel1, long hModel2);

**Description**
Adjusts a path using a cylindrical frame. Each frame in the path is adjusted so that its angle to the surface of the cylinder is the same as the angle of the first frame.

**Parameters**
long hModel1
The handle of the model that the path belongs to
long hModel2
The handle of the model that is the center of the cylinder

**Return Value**
TRUE(1), if succeed
FALSE(0), if failed

**See Also**
AdjustPathAllAsDefaultFrame()
AdjustPathAllAsMirrorFrame()
AdjustPathAllAsNearPolygonFrame()
AdjustPathAllAsNextPoint()
AdjustPathAllAsPathFrame()
long AdjustPathAllAsDefaultFrame(long hModel1, long hModel2);

**Description**
Adjusts a path using a default frame. After the first frame, each frame in a path is adjusted to arrange its orientation to the default frame so that it is the same as the orientation of the first frame to the default frame.

**Parameters**
- long hModel1
  The handle of the model that the path belongs to
- long hModel2
  The handle of the model that is the default frame.

**Return Value**
- TRUE(1), if succeed
- FALSE(0), if failed

**See Also**
- AdjustPathAllAsCylindricalFrame()
- AdjustPathAllAsMirrorFrame()
- AdjustPathAllAsNearPolygonFrame()
- AdjustPathAllAsNextPoint()
- AdjustPathAllAsPathFrame()
long AdjustPathAllAsMirrorFrame(long hModel1, long hModel2, long nNewVal1Face, long nNewVal2Pose);

**Description**
Adjusts a path using the mirroring transfer. Each frame in a path is transferred as a mirroring frame.

**Parameters**
- long hModel1: The handle of the model that the path belongs to
- long hModel2: The handle of the model that is the mirroring frame
- long nNewVal1Face:
  - 1: mirroring to X-Y plane. The sign of Z-value is changed.
  - 2: mirroring to Y-Z plane. The sign of X-value is changed.
  - 3: mirroring to Z-X plane. The sign of Y-value is changed.
- long nNewVal2Pose:
  - 0: indicates keeping the same orientation.
  - 1: indicates keeping X-Y plane.
  - 2: indicates keeping Y-Z plane.
  - 3: indicates keeping Z-X plane.
  - 4: indicates rotating a frame as 180 (degree) around the X-axis.
  - 5: indicates rotating a frame as 180 (degree) around the Y-axis.
  - 6: indicates rotating a frame as 180 (degree) around the Z-axis.

**Return Value**
- TRUE(1), if succeed
- FALSE(0), if failed

**See Also**
- AdjustPathAllAsCylindricalFrame()
- AdjustPathAllAsDefaultFrame()
- AdjustPathAllAsNearPolygonFrame()
- AdjustPathAllAsNextPoint()
- AdjustPathAllAsPathFrame()
long AdjustPathAllAsNearPolygonFrame(long hModel1, long hModel2, double dDistance, long bIsRecursive);

**Description**
Adjusts a path using a near-polygon frame. Each frame in a path is transferred to keep its orientation constant relative to a near-polygon frame.

**Parameters**
- long **hModel1**: The handle of the model that a path belongs to.
- long **hModel2**: The handle of the model that has polygons.
- double **dDistance**: Defines the distance that is used for recognizing a near-polygon.
- long **bIsRecursive**:
  - TRUE(1), searches a polygon through its child models.
  - FALSE(0), searches a polygon within a specified model.

**Return Value**
- TRUE(1), if succeed
- FALSE(0), if failed

**See Also**
- AdjustPathAllAsCylindricalFrame()
- AdjustPathAllAsDefaultFrame()
- AdjustPathAllAsMirrorFrame()
- AdjustPathAllAsNextPoint()
- AdjustPathAllAsPathFrame()
long AdjustPathAllAsNextPoint(long nNewAxisDir);

Description
Adjusts a path by forcing a specified coordinate to be directed to the next point. The last frame of a path is as same as the previous posture.

Parameters
long nNewAxisDir
0: changes X-coordinate to be directed to the next point.
1: changes Y-coordinate to be directed to the next point.
2: changes Z-coordinate to be directed to the next point.

Return Value
TRUE(1), if succeed
FALSE(0), if failed

See Also
AdjustPathAllAsCylindricalFrame()
AdjustPathAllAsDefaultFrame()
AdjustPathAllAsMirrorFrame()
AdjustPathAllAsNearPolygonFrame()
AdjustPathAllAsPathFrame()

long AdjustPathAllAsPathFrame();

Description
Adjusts a path using the nearest 3 points. Each frame in a path is defined by using the nearest 3 points. The three points define a plane and the frame is adjusted so that Z is perpendicular to this plane.

Parameters
None

Return Value
TRUE(1), if succeed
FALSE(0), if failed

See Also
AdjustPathAllAsCylindricalFrame()
AdjustPathAllAsDefaultFrame()
AdjustPathAllAsMirrorFrame()
AdjustPathAllAsNearPolygonFrame()
AdjustPathAllAsNextPoint()
long AdjustSetCondition(long nIdx1, long nNewVal1);

**Description**
Sets whether frames will be adjusted to keep a direction constant. Currently the only supported adjustment is the Z direction. When this parameter is set TRUE subsequent frames will be adjusted so that the Z direction is parallel to the Z direction of the first frame. This can be used as "Symmetry Tool".

**Parameters**
- **long nIdx1**
  Indicates the type of adjusting. The only supported value is 0.
- **long nNewVal1**
  TRUE(1), if a parameter is On.
  FALSE(0), if a parameter is Off.

**Return Value**
- TRUE(1), if a parameter is On.
- FALSE(0), if a parameter is Off.

**See Also**
- AdjustGetCondition()

long Axis6AddTail();

**Description**
Adds a new frame at the end of a path.

**Parameters**
None

**Return Value**
A new frame position. A frame handle can be obtained by GetNext() function.

**See Also**
- GetNext()
- GetData()
- SetData()
long Axis6InsertAfter(long nPos1);

**Description**
Inserts a new frame after the indicated position.

**Parameters**
long nPos1
The frame position.

**Return Value**
A new frame position. A frame handle can be obtained by GetNext() function.

**See Also**
GetNext()
GetData()
SetData()

long Axis6InsertBefore(long nPos1);

**Description**
 Inserts a new frame before the indicated position.

**Parameters**
long nPos1
The frame position.

**Return Value**
A new frame position. A frame handle can be obtained by GetNext() function.

**See Also**
GetNext()
GetData()
SetData()

void Axis6RemoveAll();

**Description**
Removes all frames.

**Parameters**
None

**Return Value**
None

**See Also**
Axis6AddTail()
Axis6RemoveAt()
void Axis6RemoveAt(long nPos1);

**Description**
Removes a frame at the indicated position.

**Parameters**
long nPos1
The frame position

**Return Value**
None

**See Also**
Axis6AddTail()
Axis6RemoveAll()

long GetCountCol();

**Description**
Gets the number of columns in a frame array. This number is 6 for AXIS6 data and 3 for points.

**Parameters**
None

**Return Value**
The number of columns

**See Also**
GetCountRow()

long GetCountRow();

**Description**
Gets the number of rows in a frame array. This number is the number of frames in the object.

**Parameters**
None

**Return Value**
The number of rows

**See Also**
GetCountCol()
double GetData(long nhAxis6, long nIdx1);

**Description**
Gets an element data of a frame.

**Parameters**
- long nhAxis6
  The frame handle
- long nIdx1
  The index of a frame with AXIS6 data.  
  0:X, 1:Y, 2:Z, 3:Rx, 4:Ry, 5:Rz

**Return Value**
Returns the specified data element

**See Also**
- GetNext()
- SetData()

long GetHeadPosition();

**Description**
Returns the head position of the frame list.

**Parameters**
None

**Return Value**
The head position of the frame list.

**See Also**
- GetNext()

long GetNext(long* nPos1);

**Description**
Returns the frame handle of the next frame in the list.

**Parameters**
- long* nPos1
  The position of a frame.

**Return Value**
The frame handle.
long* nPos1
The position of the next frame.

**See Also**
- GetHeadPosition()
void SetData(long nhAxis6, long nIdx1, double dNewValue1);

**Description**
Sets an element data of a frame.

**Parameters**
- long nhAxis6
  The frame handle
- long nIdx1
  The index of frame with AXIS6 data.
  0:X, 1:Y, 2:Z, 3:Rx, 4:Ry, 5:Rz
- double dNewValue1
  The new value of an element

**Return Value**
None

**See Also**
GetNext()
GetData()

long UpdateAxis6(long hModel1, long hShape1Axis6, long nIsUpdate);

**Description**
Loads all Shape (AXIS6) data to the frame list, if nIsUpdate flag is TRUE(1). Stores all frame-list data to the Shape (AXIS6) data, if nIsUpdate flag is FALSE(0). Call "Axis6RemoveAll()" before loading AXIS6.

**Parameters**
- long hModel1
  The model handle
- long hShape1Axis6
  The shape handle
- long nIsUpdate
  TRUE (1), Loads data to a frame list
  FALSE (0), Stores data from a frame list

**Return Value**
TRUE(1), if succeed
FALSE(0), if failed

**See Also**
Axis6RemoveAll()
class ICmdRobotSvr : public ColeDispatchDriver
long DeleteRobot(long hRobot);

**Description**
Deletes a robot object.

**Parameters**
long hRobot
The robot handle

**Return Value**
TRUE(1), if succeed
FALSE(0), if failed

**See Also**
NewRobot()  

long GetCount();

**Description**
Returns the total number of robots.

**Parameters**
None

**Return Value**
The total number of robots.

**See Also**
NewRobot()  

long GetRobotHeadPosition();

**Description**
Gets the head position of a robot list.

**Parameters**
None

**Return Value**
The head position of a robot list

**See Also**
GetRobotNext()
CString GetRobotName(long hRobot);

**Description**
Gets the robot name string.

**Parameters**
long hRobot
The handle of a robot

**Return Value**
The robot name string

**See Also**
GetRobotNext()

long GetRobotNext(long* pPos);

**Description**
Gets the robot handle.

**Parameters**
long* pPos
The position of a robot

**Return Value**
Returns a robot handle.
long* pPos
The position of the next robot

**See Also**
GetRobotHeadPosition()

LPDISPATCH NewCmdAutoPlace(long hRobot);

**Description**
Gets an ICmdAutoPlace object.

**Parameters**
long hRobot
A robot handle

**Return Value**
Returns a dispatch pointer.

**See Also**
NewCmdMotoPosition()
LPDISPATCH NewCmdMotoPosition();

Description
Gets an ICmdMotoPosition object.

Parameters
None

Return Value
Returns a dispatch pointer.

See Also
NewCmdAutoPlace()

LPDISPATCH NewCmdMotoRobot(long hRobot);

Description
Gets an ICmdMotoRobot object.

Parameters
long hRobot
A robot handle

Return Value
Returns a dispatch pointer.

See Also
NewCmdMotoPosition()

LPDISPATCH NewCmdMotoRobotTool();

Description
Gets an ICmdMotoRobotTool object.

Parameters
None

Return Value
Returns a dispatch pointer.

See Also
NewCmdMotoPosition()
NewCmdMotoRobot()
long NewRobot(LPCTSTR strName, LPCTSTR strRobotPath);

**Description**
Creates a new robot object.

**Parameters**
- LPCTSTR strName
  The name of a robot. This name should be unique for the model name.
- LPCTSTR strRobotPath
  The robot path name. This folder will hold the robot information.

**Return Value**
The handle of a robot if successful, and NULL otherwise.

**See Also**
DeleteRobot()

void SetRobotName(long hRobot, LPCTSTR lpszNewValue);

**Description**
Changes the robot name. This function can be used after creating a robot object but it should be avoided if possible. When a robot object is created with the name "Robot1" all its links are also named "Robot1". If SetRobotName() is used to rename the robot to "Robot2" its links will still be named "Robot1". This may cause problems.

**Parameters**
- long hRobot
  The robot handle
- LPCTSTR lpszNewValue
  The new robot name

**Return Value**
None

**See Also**
NewRobot()
class ICmdModelIO : public COleDispatchDriver
long Create(long nType);

Description
Creates an IO module object. The type of IO module is either an input type or an output type.

Parameters
long nType
0: input type
1: output type

Return Value
Returns a handle of an IO object if successful, and NULL otherwise.

See Also
Delete()
SetItemText()
GetType()

void Delete(long hMSimIO);

Description
Delete IO module object.

Parameters
long hMSimIO
IO module handle

Return Value
None

See Also
Create()
GetHeadPosition()
GetItemText()
long GetBit(long hMdlIo, long nIdx1);

**Description**
Gets a data bit.

**Parameters**
- long hMdlIo
  IO module handle
- long nIdx1
  Index number of IO bit address

**Return Value**
- TRUE(1), if the IO bit is On.
- FALSE(0), if the IO bit is Off.

**See Also**
SetBit()

long GetByte(long hMdlIo, long nIdx1);

**Description**
Gets a data byte (8 bits).

**Parameters**
- long hMdlIo
  IO module handle
- long nIdx1
  Index number of IO byte address

**Return Value**
Returns 8 bits value, 0 to 255.

**See Also**
SetByte()
long GetCount();

**Description**
Gets the count of the number of IO modules.

**Parameters**
None

**Return Value**
The number of IO modules

**See Also**
GetHeadPosition()
GetNext()

long GetHeadPosition();

**Description**
Gets the head-position of IO module list.

**Parameters**
None

**Return Value**
The head-position of IO module list

**See Also**
GetNext()

CString GetItemText(long hMdlIo);

**Description**
Gets the item text of an IO module.

**Parameters**
long hMdlIo
IO module handle

**Return Value**
Returns the item text of a IO module

**See Also**
SetItemText()
long GetMaxNum(long hMdlIo);

**Description**
Gets the max number of bits in IO module.

**Parameters**
- long hMdlIo
  - IO module handle

**Return Value**
Returns the max number of bits in IO module

**See Also**
- SetMaxNum()

long GetNext(long* pnPos1);

**Description**
Gets an IO module handle in IO module list.

**Parameters**
- long* pnPos1
  - IO module position

**Return Value**
Returns IO module handle.
- long* pnPos1
  - The next IO module position

**See Also**
- GetHeadPosition()

long GetNumLinkFrom(long hMSimIO, long nIdx1);

**Description**
Gets the link number of an input connection.

**Parameters**
- long hMSimIO
  - IO module handle
- long nIdx1
  - Index number of IO byte address

**Return Value**
Returns the link number.

**See Also**
- GetNumLinkTo()
long GetNumLinkTo(long hMSimIO, long nIdx1);

**Description**
Gets the link number of an output connection.

**Parameters**
- long hMSimIO
  IO module handle
- long nIdx1
  Index number of IO byte address

**Return Value**
Returns the link number.

**See Also**
GetNumLinkFrom()

long GetType(long hMdlIo);

**Description**
Gets the type of an IO module. The type of an IO module is either an input type or an output type.

**Parameters**
- long hMdlIo
  IO module handle

**Return Value**
Returns a type of IO module.
- 0: input type
- 1: output type

**See Also**
Create()
SetItemText()
SetType()

long IsExist();

**Description**
Gets the exist flag of IO module

**Parameters**
None

**Return Value**
TRUE(1), if IO modules exist.
FALSE(0), if IO modules do not exist.

**See Also**
GetCount()
4-96 OLE

void SetBit(long hMdlIo, long nIdx1, long nNewValue);

**Description**
Sets a data bit.

**Parameters**
- long hMdlIo
  - IO module handle
- long nIdx1
  - Index number of IO bit address
- long nNewValue
  - TRUE(1), turns an IO bit On.
  - FALSE(0), turns an IO bit Off.

**Return Value**
None

**See Also**
- GetBit()

void SetByte(long hMdlIo, long nIdx1, long nNewValue);

**Description**
Sets a data byte (8 bits).

**Parameters**
- long hMdlIo
  - IO module handle
- long nIdx1
  - Index number of IO byte address
- long nNewValue
  - 8 bits value, 0 to 255.

**Return Value**
None

**See Also**
- GetByte()
void SetItemText(long hMdlIo, LPCTSTR lpszNewValue);

Description
Sets the item text of an IO module.

Parameters
long hMdlIo
IO module handle
LPCTSTR lpszNewValue
Item text of an IO module

Return Value
None

See Also
GetItemText()

void SetMaxNum(long hMdlIo, long nNewValue);

Description
Sets the max number of bits in IO module.

Parameters
long hMdlIo
IO module handle
long nNewValue
Max number of bits in IO module

Return Value
None

See Also
GetMaxNum()
void SetType(long hMdlIo, long nNewValue);

**Description**
Sets the type of an IO module. The type of an IO module is either input or output.

**Parameters**
- long hMdlIo
  - IO module handle
- long nNewValue
  - 0: input type
  - 1: output type

**Return Value**
None

**See Also**
- Create()
- SetItemText()
- GetType()
class ICmdMIOBitConnect : public COleDispatchDriver
long Add(long hModel1, long hMSimIO1, long nIdx1, long hModel2, long hMSimIO2, long nIdx2);

Description
Adds an IO module connection from an output module to an input module.

Parameters
long hModel1
The handle of the model of an output IO module
long hMSimIO1
The handle of an output IO module
long nIdx1
The index number of an output IO address.
long hModel2
The Model handle of an input IO module
long hMSimIO2
The handle of an input IO module
long nIdx2
The index number of an input IO address.

Return Value
TRUE(1), if succeed
FALSE(0), if failed

See Also
Delete()
long Delete(long hModel1, long hMSimIO1, long nIdx1, long hModel2, long hMSimIO2, long nIdx2);

**Description**
Deletes an IO module connection.

**Parameters**
- `long hModel1`: The model handle of an output IO module
- `long hMSimIO1`: The handle of an output IO module
- `long nIdx1`: The index number of an output IO address.
- `long hModel2`: The model handle of an input IO module
- `long hMSimIO2`: The handle of an input IO module
- `long nIdx2`: The index number of an input IO address.

**Return Value**
- TRUE(1), if succeed
- FALSE(0), if failed

**See Also**
- Add()
- DeleteAll();

long DeleteAll();

**Description**
Deletes all IO module connections.

**Parameters**
None

**Return Value**
- TRUE(1), if succeed
- FALSE(0), if failed

**See Also**
- Add()
- Delete()
long Node1GetCount();

Description
Gets the count of output nodes

Parameters
None

Return Value
The count of output connection nodes

See Also
Node1GetHeadPosition()
Node1GetNext()

long Node1GetHeadPosition();

Description
Gets the head position of the output nodes list.

Parameters
None

Return Value
The head position of the output nodes list

See Also
Node1GetNext()

long Node1GetModelHandle(long hNode1);

Description
Get a model handle of the node.

Parameters
long hNode1
The node handle

Return Value
Returns a model handle of the node.

See Also
Node1GetMSimIOHandle()
long Node1GetMSimIOHandle(long hNode1);

**Description**
Get an IO module handle of the node.

**Parameters**
- long hNode1
  The node handle

**Return Value**
Returns an IO module handle of the node.

**See Also**
Node1GetModelHandle()

long Node1GetNext(long* pnPos1);

**Description**
Gets the output node handle.

**Parameters**
- long* pnPos1
  The position of a Node1 list

**Return Value**
Returns the output node handle
- long* pnPos1
  The next position of a Node1 list

**See Also**
Node1GetHeadPosition()

long Node2GetCount(long hNode1);

**Description**
Gets the connection count of the node.

**Parameters**
- long hNode1
  The node handle

**Return Value**
Returns the connection count of the node.

**See Also**
Node2GetHeadPosition()
Node2GetNext()
long Node2GetHeadPosition(long hNode1);

**Description**
Gets the head position of the connection list.

**Parameters**
long hNode1
The node handle

**Return Value**
Returns the head position of the connection list.

**See Also**
Node2GetNext()
Node2GetCount()

long Node2GetIndex1(long hNode2);

**Description**
Gets the index of the output module.

**Parameters**
long hNode2
The Node2 handle

**Return Value**
Returns the index of the output module.

**See Also**
Node2GetIndex2()

long Node2GetIndex2(long hNode2);

**Description**
Gets the index of the input module.

**Parameters**
long hNode2
The Node2 handle

**Return Value**
Returns the index of the input module.

**See Also**
Node2GetIndex1()
long Node2GetModelHandle(long hNode2);

Description
Get the model handle of the node.

Parameters
long hNode2
The node handle

Return Value
Returns a model handle of the node.

See Also
Node2GetMSimIOHandle()

long Node2GetMSimIOHandle(long hNode2);

Description
Get an IO module handle of the node.

Parameters
long hNode2
The node handle

Return Value
Returns an IO module handle of the node.

See Also
Node2GetModelHandle()

long Node2GetNext(long hNode1, long* pnPos1);

Description
Gets the connection handle of Node2. Node2 is the list of input nodes connected to the current output Node1.

Parameters
long hNode1
The node handle
long* pnPos1
The Node2 position

Return Value
Returns the head position of the connection list.
long* pnPos1
The next Node2 position

See Also
Node2GetHeadPosition ()
Node2GetCount ()
long OnUpdate();

*Description*
Updates the connection status of all IO nodes.

*Parameters*
None

*Return Value*
TRUE(1), if succeed
FALSE(0), if failed

*See Also*
OnUpdateAll()

long OnUpdateAll();

*Description*
Updates the connection status of all nodes. Also deletes connection nodes if the input terminals were deleted.

*Parameters*
None

*Return Value*
TRUE(1), if succeed
FALSE(0), if failed

*See Also*
OnUpdate()
class ICmdMotoRobotMemory : public COleDispatchDriver
long FileStoring(LPCTSTR strFile, long fIsStoring);

**Description**
Saves or loads a robot file. The current supported files are "TOOL" and "UFRAME".

**Parameters**
- LPCTSTR strFile
  "TOOL" or "UFRAME"
- long fIsStoring
  TRUE(1), for saving.
  FALSE(0), for loading.

**Return Value**
- TRUE(1), if succeed
- FALSE(0), if failed

**See Also**
- long ToolGetCount();
  **Description**
  Gets the count of tool slots.
  **Parameters**
  None
  **Return Value**
  Returns the count of tool slots.
  **See Also**
  ToolGetHeadPosition()
  ToolGetNext()

long ToolGetHeadPosition();

**Description**
Gets the head position of the tool list.
**Parameters**
None
**Return Value**
Returns the head position of the tool list.
**See Also**
ToolGetNext()
long ToolGetNext(long* pnPos1);

**Description**
Gets a tool handle of the tool list.

**Parameters**
long* pnPos1
The position of the tool list

**Return Value**
Returns a tool handle of the tool list.

long* pnPos1
The position of the next tool list

**See Also**
ToolGetHeadPosition()

long ToolGetProperty(long nTool1, LPDISPATCH lpDispatch);

**Description**
Gets the tool property into the ICmdMotoRobotTool object.

**Parameters**
long nTool1
Tool handle
LPDISPATCH lpDispatch
ICmdMotoRobotTool object.

**Return Value**
TRUE(1), if succeed
FALSE(0), if failed

**See Also**
ToolSetProperty()
long ToolSetProperty(long nTool1, LPDISPATCH lpdispatch);

**Description**
Sets the tool property into the ICmdMotoRobotTool object.

**Parameters**
- long nTool1
  Tool handle
- LPDISPATCH lpdispatch
  ICmdMotoRobotTool object.

**Return Value**
- TRUE(1), if succeed
- FALSE(0), if failed

**See Also**
- ToolGetProperty()
class ICmdMotoRobotTool : public COleDispatchDriver
CString GetItemText();

Description
Gets the tool name.
Parameters
None
Return Value
Returns the tool name.
See Also
SetItemText()

double GetMass();

Description
Gets the tool mass
Parameters
None
Return Value
Returns the tool mass. (kg)
See Also
SetMass()

long GetToolNumber();

Description
Gets the tool number.
Parameters
None
Return Value
Returns the tool number.
See Also
SetToolNumber()
double GetXyzMxyzOfMass(long nIdx1);

**Description**  
Gets an AXIS6 element of the mass attribute.

**Parameters**  
long nIdx1  
Element index of AXIS6.  
0: X, 1:Y, 2:Z, 3:Rx, 4:Ry, 5:Rz

**Return Value**  
Returns an element value.  
(mm) for X, Y, Z  
(rad) for Rx, Ry, Rz

**See Also**  
SetXyzMxyzOfMass()

double GetXyzRxyz(long nIdx1);

**Description**  
Gets an AXIS6 element of the TCP (tool center point).

**Parameters**  
long nIdx1  
Element index of AXIS6.  
0: X, 1:Y, 2:Z, 3:Rx, 4:Ry, 5:Rz

**Return Value**  
Returns an element value.  
(mm) for X, Y, Z  
(rad) for Rx, Ry, Rz

**See Also**  
SetXyzRxyz()

void SetItemText(LPCTSTR strNewValue);

**Description**  
Sets the tool name.

**Parameters**  
LPCTSTR strNewValue  
Tool name.

**Return Value**  
None

**See Also**  
GetItemText()
void SetMass(double dNewValue);

**Description**
Sets the tool mass

**Parameters**
double dNewValue
Tool mass (kg)

**Return Value**
None

**See Also**
SetMass()

void SetToolNumber(long nNewValue);

**Description**
Sets the tool number.

**Parameters**
long nNewValue
Tool number

**Return Value**
None

**See Also**
GetToolNumber()

void SetXyzMxyzOfMass(long nIdx1, double dNewValue);

**Description**
Sets an AXIS6 element of the mass attribute.

**Parameters**
long nIdx1
Element index of AXIS6.
0: X, 1:Y, 2:Z, 3:Rx, 4:Ry, 5:Rz
double dNewValue
Element value.
(mm) for X, Y, Z
(rad) for Rx, Ry, Rz

**Return Value**
None

**See Also**
GetXyzMxyzOfMass()
void SetXyzRxyz(long nIdx1, double dNewValue);

**Description**
Sets an AXIS6 element of the TCP (tool center point).

**Parameters**
- long nIdx1
  Element index of AXIS6
  0: X, 1:Y, 2:Z, 3:Rx, 4:Ry, 5:Rz
- double dNewValue
  Element value.
  (mm) for X, Y, Z
  (rad) for Rx, Ry, Rz

**Return Value**
None

**See Also**
GetXyzRxyz ()
class ICmdMotoRobotSystem : public COleDispatchDriver

long GetJointMaxPulse(long nIdx1);

Description
Gets the upper limit number of a joint pulse.

Parameters
long nIdx1
Index of a joint. (starts with 0)

Return Value
The upper limit number of a joint pulse

See Also
GetJointMinPulse()
SetJointMaxPulse()

long GetJointMinPulse(long nIdx1);

Description
Gets the lower limit number of a joint pulse.

Parameters
long nIdx1
Index of a joint. (starts with 0)

Return Value
The lower limit number of a joint pulse

See Also
GetJointMaxPulse()
SetJointMinPulse()
long GetJointType(long nIdx1);

Description
Gets the joint type (Rotating type or Sliding type).

Parameters
long nIdx1
Index of a joint. (starts with 0)

Return Value
the joint type
0: X-axis sliding type
1: Y-axis sliding type
2: Z-axis sliding type
3: Rx-axis rotating type
4: Ry-axis rotating type
5: Rz-axis rotating type

See Also
GetJointYaXb()
SetJointType()

long GetJointYaXb(long nIdx1, double* pdA, double* pdB);

Description
Gets parameters of joint formula. \( Y = a \times X + b \); Here \( X \) is pulse value, and \( Y \) is an angle (rad) or a length (mm).

Parameters
long nIdx1
Index of a joint. (starts with 0)
double* pdA
parameter "a"
double* pdB
parameter "b"

Return Value
TRUE(1), if succeed
FALSE(0), if failed

See Also
GetJointYaXb()
SetJointYaXb()
void SetJointMaxPulse(long nIdx1, long nNewValue1);

**Description**
Sets the upper limit number of a joint pulse. This function may not be implemented for a particular robot model. If the function does not work correctly it will be necessary to modify the original robot data directly.

**Parameters**
- long nIdx1
  Index of a joint. (starts with 0)
- long nNewValue1
  The limit number of the joint pulse

**Return Value**
None

**See Also**
- GetJointMinPulse()
- GetJointMaxPulse()

void SetJointMinPulse(long nIdx1, long nNewValue1);

**Description**
Sets the lower limit number of a joint pulse. This function may not be implemented for a particular robot model. If the function does not work correctly it will be necessary to modify the original robot data directly.

**Parameters**
- long nIdx1
  Index of a joint. (starts with 0)
- long nNewValue1
  The limit number of a joint pulse

**Return Value**
None

**See Also**
- GetJointMaxPulse()
- GetJointMinPulse()
void SetJointType(long nIdx1, long nNewValue1);

**Description**
Sets joint type (Rotating type or Sliding type). This function may not be implemented for a particular robot model. If the function does not work correctly, modify the original robot data directly.

**Parameters**
- long nIdx1: Index of a joint. (starts with 0)
- long nNewValue1: the joint type
  - 0: X-axis sliding type
  - 1: Y-axis sliding type
  - 2: Z-axis sliding type
  - 3: Rx-axis rotating type
  - 4: Ry-axis rotating type
  - 5: Rz-axis rotating type

**Return Value**
None

**See Also**
- GetJointYaXb()
- GetJointType()

void SetJointYaXb(long nIdx1, double dNewValue1, double dNewValue2);

**Description**
Sets parameters of joint formula. Y = a*X + b; Here X is pulse value, and Y is an angle (rad) or a length (mm).

**Parameters**
- long nIdx1: Index of a joint. (starts with 0)
- double dNewValue1: parameter "a"
- double dNewValue2: parameter "b"

**Return Value**
None

**See Also**
- GetJointYaXb()
class ICmdModelType : public COleDispatchDriver
long GetCount();

Description
Gets the count of the model type list.
Parameters
None
Return Value
Returns the count of the model type list
See Also
GetHeadPosition()
GetNext()

long GetHeadPosition();

Description
Gets the head position of the model type list.
Parameters
None
Return Value
Returns the head position.
See Also
GetNext()

CString GetItemText();

Description
Gets the item text of the model type list.
Parameters
None
Return Value
Returns the item text of the model type list.
See Also
GetNext()
GetHeadPosition()
long GetNext(long* pPos1);

**Description**
Gets the model handle of a type model.

**Parameters**
long* pPos1
The position of the model

**Return Value**
Returns a model handle.

long* pPos1
The position of the next model

**See Also**
GetHeadPosition()
Index

A
About This Document 1-1
Add 4-99
AddModel 4-8
AdjustAfterMulMatrix 4-76
AdjustAfterMulMatrixAll 4-76
AdjustGetCondition 4-77
AdjustPathAllAsCylindricalFrame 4-77
AdjustPathAllAsDefaultFrame 4-78
AdjustPathAllAsMirrorFrame 4-79
AdjustPathAllAsNearPolygonFrame 4-80
AdjustPathAllAsNextPoint 4-81
AdjustPathAllAsPathFrame 4-81
AdjustSetCondition 4-82
Angle 3-2
ANSI/RIA 2-1
Asynchronous Simulation 3-3
AttrGetCount 4-61
AttrGetDefaultRect 4-61
AttrGetHandle 4-56
AttrGetHeadPosition 4-62
AttrGetItemText 4-62
AttrGetNext 4-63
AttrSetDefaultRect 4-63
AttrSetHandle 4-56
AXIS6 model 3-3
Axis6AddTail 4-82
Axis6InsertAfter 4-83
Axis6InsertBefore 4-83
Axis6RemoveAll 4-83
Axis6RemoveAt 4-84
CalcQuickCycleTime 4-6
Check 4-9
CloseCell 4-2
ConvertMatrix 4-48
CopyLine2Axis6 4-27
Create 4-91
CreateDispatch 3-1
Customer Service 1-2
Delete 4-48, 4-91, 4-100
DeleteAll 4-100
DeleteAllLines 4-28
DeleteAllShapes 4-28
DeleteRobot 4-87
DeleteShape 4-28
DelModel 4-9
F
FileStore 4-2
FileStoring 4-106
G
General Safeguarding Tips 2-2
GetAttributeDoubleByIndex 4-29
GetAttributeLongByIndex 4-29
GetAxis6 4-24
GetBit 4-92
GetByte 4-92
GetCheckAll 4-9
GetCheckAllCombinations 4-10
GetChildItem 4-49
GetCollision 4-10
GetCount 4-11, 4-87, 4-93, 4-117
GetCountCol 4-84
GetCountRow 4-84
GetCurrentJobLine 4-64
GetCurrentJobName 4-64
GetCurrentJobStep 4-64
GetCurrentToolHandle 4-65
GetCycleTime 4-6, 4-18, 4-65
GetData 4-85
GetElement 4-24
GetFileName 4-3, 4-30
GetHandle 4-24, 4-30, 4-49, 4-65
GetHeadPosition 4-11, 4-85, 4-93, 4-117
GetHeartBeat 4-19
GetHideSee 4-30
GetHold 4-19
GetInFilePath 4-6
GetItemText 4-93, 4-109, 4-117
GetJobName 4-19
GetJointMaxPulse 4-113
GetJointMinPulse 4-113
GetJointType 4-114
GetJointYaXb 4-114
N
New 4-51
NewCmdAutoPlace 4-88
NewCmdCollision 4-51
NewCmdGeneralOp 4-3
NewCmdMatrix 4-52
NewCmdMIOBitConnect 4-52
NewCmdModel 4-52
NewCmdModelIO 4-53
NewCmdModelSvr 4-4
NewCmdModelType 4-53
NewCmdMotoJob 4-70
NewCmdMotoPosition 4-89
NewCmdMotoRobot 4-89
NewCmdMotoRobotMemory 4-70
NewCmdMotoRobotSystem 4-70
NewCmdMotoRobotTool 4-89
NewCmdProcessAngle 4-53
NewCmdRobotSvr 4-4
NewRobot 4-90
NewShape 4-36
Node1GetCount 4-101
Node1GetHeadPosition 4-101
Node1GetModelHandle 4-101
Node1GetMSimIOHandle 4-102
Node2GetMSimIOHandle 4-104
Node2GetNext 4-104
Normalize 4-25
O
OnPlay 4-21
OnUpdate 4-105
OnUpdateAll 4-105
Operation Safety 2-4
P
Playback Simulation 3-3
Process Angle Object 3-4
Programming Safety 2-3
PrpGetPositionType 4-60
PrpSetPositionType 4-60
Pulse 3-2
R
RbModuleGetCount 4-71
RbModuleGetHeadPosition 4-71
RbModuleGetItemText 4-71
RbModuleGetJointCount 4-72
RbModuleGetNext 4-72
RbModuleJointGetAxisBits 4-72
Recalc 4-16, 4-54
Rectangle Data 3-2
Reference to Other Documentation 1-2
Refreshview 4-37
RefreshWindow 4-4
ReleaseDispatch 3-1
RelocateAllShapes 4-37
RelocateModel 4-54
Robot INF Script 3-2
Robot Job Handling 3-2
Robot Object 3-1
Robot TCP Posture 3-2
Robot Tool data 3-1
Robotic Industries Association 2-1
S
SaveFile 4-37
setAttributeDoubleByIndex 4-38
setAttributeLongByIndex 4-38
SetAxis6 4-25
SetBit 4-96
SetByte 4-96
SetCheckAll 4-16
SetCheckAllCombinations 4-17
SetCurrentJobLine 4-73
SetCurrentJobName 4-73
SetCurrentToolHandle 4-73
SetCycleTime 4-7, 4-22, 4-74
SetData 4-86
SetElement 4-26
SetFileName 4-5, 4-39
SetHandle 4-39
SetHeartBeat 4-22
SetHideSee 4-39
SetHold 4-23
SetInfFilePath 4-8
SetItemText 4-97, 4-110
SetJobName 4-23
SetJointMaxPulse 4-115
Index

SetJointMinPulse 4-115
SetJointType 4-116
SetJointYaXb 4-116
SetMass 4-111
SetMatrix 4-40
SetMaxNum 4-97
SetMdlFilePath 4-74
SetModelType 4-40
SetMotionModuleCode 4-75
SetName 4-41, 4-55
SetParent 4-41
setPosition 4-42
SetPowerStatus 4-75
SetRobotName 4-23, 4-90
SetShapeColor 4-42
SetShapeData 4-43
SetShapeData6 4-43
SetShapeName 4-44
SetToolNumber 4-111
SetType 4-55, 4-98
SetUpdateColorForAllShapes 4-17
SetXyzMxyzOfMass 4-111
SetXyzRxyz 4-112
Shape Object 3-3
ShapeCopyRow 4-44
ShapeGetCount 4-45
ShapeGetHeadPosition 4-45
ShapeGetNext 4-45
ShapeGetRowColMax 4-46
ShapeGetType 4-46
ShapeMerge 4-47
ShapeRefreshView 4-47
ShowWindow 4-5
Standard Conventions 2-1
Sub 4-26
Synchronous Simulation 3-3
System Requirements 1-2
T
ToolGetCount 4-106
ToolGetHeadPosition 4-106
ToolGetNext 4-107
ToolGetProperty 4-107
ToolSetProperty 4-108
Trans 4-27
U
UpdateAxis6 4-86
UpdateColor 4-18
Use Cylindrical Frame 3-4
Use Default Frame 3-4
Use Near Polygon 3-4
Use Path Frame 3-4