

# SANDIA REPORT

SAND98-2770

Unlimited Release

Printed December 1998

## LandScape Command Set

# Local Area Network Distributed Supervisory Control and Programming Environment

Ross L. Burchard and Daniel E. Small

Prepared by  
Sandia National Laboratories  
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,  
a Lockheed Martin Company, for the United States Department of  
Energy under Contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.



**Sandia National Laboratories**

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

**NOTICE:** This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from  
Office of Scientific and Technical Information  
P.O. Box 62  
Oak Ridge, TN 37831

Prices available from (615) 576-8401, FTS 626-8401

Available to the public from  
National Technical Information Service  
U.S. Department of Commerce  
5285 Port Royal Rd  
Springfield, VA 22161

NTIS price codes  
Printed copy: A03  
Microfiche copy: A01



**SAND98-2770  
Unlimited Release  
Printed December 1998**

# **LandScape Command Set**

**Local Area Network  
Distributed Supervisory Control  
And Programming Environment**

**Ross L. Burchard  
Daniel E. Small  
Intelligent Systems and Robotics Center**

**Sandia National Laboratories  
P.O. Box 5800  
Albuquerque, NM 87185**

## **Abstract**

This paper presents the Local Area Network Distributed Supervisory Control and Programming Environment (LandScape) commands set that provides a Generic Device Subsystem Application Programmers Interface (API). These commands are implemented using the Common Object Request Broker Architecture (CORBA) specification with Orbix from Iona Technologies.

---

---

## TABLE OF CONTENTS

---

<b>1.0 INTRODUCTION</b> .....	<b>1</b>
<b>2.0 CLIENT IDL</b> .....	<b>2</b>
2.1 NON-BLOCKING AND STATE-NEUTRAL .....	3
2.2 BLOCKING AND STATE-NEUTRAL .....	3
<b>3.0 SERVER IDL</b> .....	<b>3</b>
3.1 CONNECT/POWER AND SINGLE POC COMMANDS .....	3
3.1.1 Blocking and Unlocked State.....	4
3.1.2 Blocking and Locked State.....	4
3.1.3 Blocking and Active State.....	4
3.2 SET COMMANDS.....	4
3.2.1 Blocking and Locked State.....	4
3.3 GET COMMANDS .....	6
3.3.1 Blocking and State Neutral.....	6
3.4 SINGLE MOTION COMMANDS .....	7
3.4.1 Non-Blocking and Active State.....	7
3.5 PATH MOTION COMMANDS.....	9
3.5.1 Blocking and Active State.....	9
3.5.2 Non-Blocking and Active State.....	11
3.6 ASYNCHRONOUS INTERRUPTION COMMANDS.....	11
3.6.1 Blocking and Moving State .....	11
3.6.2 Blocking and Paused or Locked State.....	11
3.7 SENSOR DEPENDENT COMMANDS.....	12
3.7.1 Blocking and Moving State .....	12
3.7.2 Non-Blocking and Active State.....	12
3.7.3 Blocking and Locked State.....	14
3.8 ENDEFFECTOR COMMANDS.....	14
3.8.1 Blocking and State Independent .....	14
<b>4.0 DIGITAL_IO IDL</b> .....	<b>15</b>
4.1 Blocking and State Independent.....	15
<b>5.0 RETURN CODES</b> .....	<b>16</b>
<b>6.0 PARAMETER CONSTANTS</b> .....	<b>17</b>

### 1.0 Introduction

LandScape commands are divided into “blocking and “non-blocking” commands. In addition, the following states as shown in Figure 1.1 are defined for the command set: *Idle*, *Unlocked*, *Locked*, and *Active*. Each LandScape command is valid and may be called while the machine or device is in the appropriate state. The commands are also divided into the following categories: client, server, digital I/O, return codes and parameter constants to better identify the capabilities and proper usage.

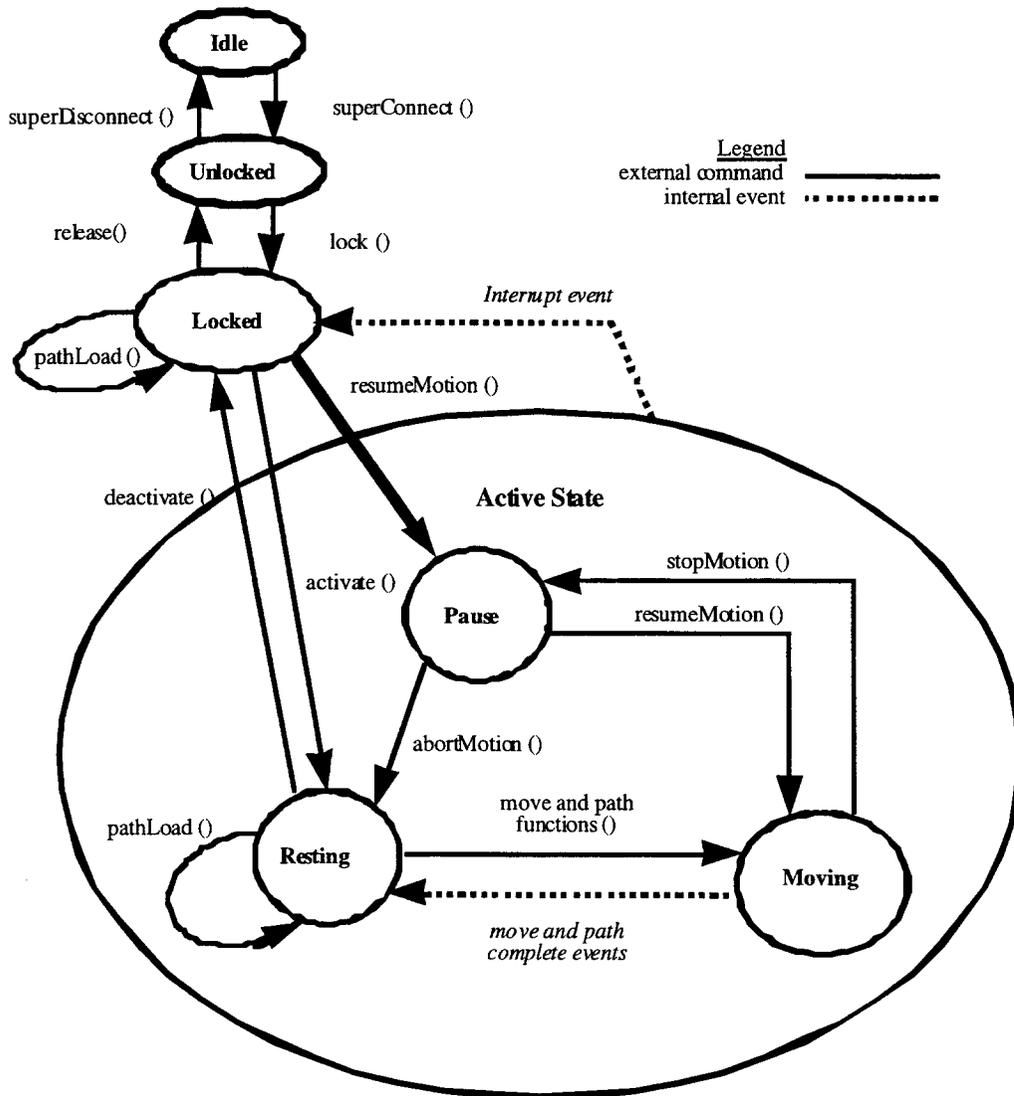


Figure 1.1 Command Transition Diagram

## 2.0 Client IDL

These are the event handlers for non-blocking commands on the server.

### 2.1 Non-blocking and State-independent

#### **oneway clientAlarm (in string eventID, in sequence string data)**

This is used for error or alarm conditions on the server. The server may at any time execute an Alarm. This alarm is Non-Blocking and requires no response or acknowledgment from the client.

#### **oneway clientEvent (in string eventID, in sequence string data)**

This is used for event conditions on the server. The server may execute an event after it has entered a busy/working state of a Non-Blocking command. This event is Non-Blocking and requires no response or acknowledgment from the client.

### 2.2 Blocking and State-independent

A disconnect handler routine will allow certain shutdown actions to be performed automatically any time a client disconnects with the subsystem (either intentionally or due to error conditions) The default behavior will be to issue a deactivate command and a release command.

#### **retVal clientAlarmRSVP (in string eventID, inout sequence string data)**

This is used for alarm conditions on the server. The server may execute an alarm at any time. This alarm is Blocking and requires a response (RSVP) from the client.

#### **retVal clientEventRSVP (in string eventID, inout sequence string data)**

This is used for event conditions on the server. The server may execute an event after it has entered a busy/working state of a Non-Blocking command. This event is Blocking and requires a response (RSVP) from the client.

## 3.0 Server IDL

This section describes the LandScape commands supported on the server.

### 3.1 Connect/Power and Single POC Commands

This section presents device connect and power on/off commands as well as single point of control (POC) commands.

#### 3.1.1 Blocking and Unlocked State

These commands are only valid only from the *Unlocked State*.

##### **retVal lock ()**

Give the supervisor exclusive REMOTE control over the subsystem, *no parameters*.

#### 3.1.2 Blocking and Locked State

These commands are only valid only from the *Locked State*.

Note: When a subsystem is booted, its default state is “Idle”.

##### **retVal release ()**

Give the subsystem exclusive LOCAL control over itself, *no parameters*.

##### **retVal activate ()**

Place the transport device in an active state such that it can be physically commanded to move, *no parameters*.

#### 3.1.3 Blocking and Active State

These commands are only valid only from the *Active State*.

##### **retVal deactivate ()**

Place the transport device in an inactive state such that it will NOT physically move if commanded to do so, *no parameters*.

### 3.2 Set Commands

This section presents device set-up and configuration commands.

#### 3.2.1 Blocking and Locked State

##### **retVal setSpeed (in double <speed>, in SpeedType <attribute>)**

---

Set the absolute speed for the robot, *two parameters*:

<speed> converted to double and its meaning depends upon <attribute>.

<attribute> one of the following TYPEDEFF'ed keywords which defines the type of speed being set.

PERCENT\_SPEED (percent of max speed applied globally to both joint and Cartesian moves).

LINEAR\_SPEED (absolute linear speed applied to Cartesian moves and interpreted according to how the last SetUnits command specified the units, such as inches/second or meters/second).

ANGULAR\_SPEED (absolute angular speed applied to Cartesian moves and interpreted according to how the last SetUnits command specified the units, such as degrees/second or radians/sec).

#### **retVal setAcceleration (in double <accel>, in long <attribute>)**

Set the absolute acceleration for the robot, *two parameters*:

<accel> converted to double and its meaning depends upon <attribute>

<attribute> one of the following enumerated keywords which defines the type of accel being set.

PERCENT\_ACCEL (percent of max accel applied globally to both joint and Cartesian moves, constant rate).

LINEAR\_ACCEL (absolute linear accel applied to Cartesian moves and interpreted according to how the last SetUnits command specified the units, such as inches/second<sup>2</sup> or meters/second<sup>2</sup>).

ANGULAR\_ACCEL (absolute angular accel applied to Cartesian moves and interpreted according to how the last SetUnits command specified the units, such as degrees/second<sup>2</sup> or radians/sec<sup>2</sup>).

#### **retVal setToolOffset (in double <offset\_array[6]>)**

Set the tool length offsets for the robot, *six parameters*:

<offset\_array> defined as: <x>, <y>, <z>, <roll>, <pitch>, <yaw> converted to double, units depend upon last SetUnits command executed (inches, degrees, meters, radians, etc.).

#### **retVal setWorldCartesianFrame (in double <x> <y> <z> <roll> <pitch> <yaw>)**

Set the offsets from the robot's base frame to the world frame., *six parameters*:

double <x>, <y>, <z>, <roll>, <pitch>, <yaw>

### 3.3 Get Commands

This section presents information retrieval commands.

#### 3.3.1 Blocking and State Neutral

**retVal getUnits (out string linear\_units, out string angular\_units)**

Returns the units that the machine is speaking in.

<linear\_units> inches/feet/meters/centimeters/millimeters.

<angular\_units> degrees/radians.

**retVal getVelocities (out double percent\_velocity, out double world\_linear\_velocity, out double world\_angular\_velocity Sequence double joint\_velocities)**

Returns current velocity settings for percentage of max velocity, world angular, world joint and individual joint velocities.

**retVal getAccelerations (out double percent\_accel, out double world\_linear\_accel, out double world\_angular\_accel, out Sequence double joint\_accels)**

Returns current velocity settings for percentage of max accelerations, world angular, world joint and individual joint accelerations.

**retVal getMaxVelocities (out double max\_world\_linear\_velocity, out double max\_world\_angular\_velocity, out double Sequence double max\_joint\_velocities)**

Returns maximum velocity settings for world angular, world linear, and world joint velocities.

**retVal getMaxAccelerations (out double max\_world\_linear\_accel, out double max\_world\_angular\_accel, out double sequence double max\_joint\_accels)**

Returns maximum acceleration settings for world angular, world linear, and world joint acceleration.

**retVal getToolOffsets (out double offset\_array[6])**

Returns the 6 Cartesian offsets of the robot from it's Cartesian base frame to the tool tip.

**retVal getRobotPosition (out sequence double current\_joint\_position, out double current\_world\_position[6])**

<current\_joint\_position> is the sequence of doubles which represents the values of each joint starting with joint 1 and going to joint N.

<current\_world\_position[6]> is the double array of the current Cartesian position of the robot.

**retVal getMotionControlState (out string motionControlState, out string controlMode)**

<motionControlState> ASCII string which describes the current trajectory state of the robot. The suggested states are:

running\_trajectory  
robot\_paused  
robot\_stopped\_on\_error  
robot\_stopped\_on\_command

<controlMode> string which describes the current state of the controller. The suggested states are:

power\_on  
power\_off  
controller\_error, etc.

**retVal getTagId (out string pathName, out string tagID)**

<pathName> string which holds the name of the current path.

<tagID> string which holds the last tag name that the robot got to.

### 3.4 Single Motion Commands

This section presents commands that perform a single device motion.

#### 3.4.1 Non-Blocking and Active State

These commands are valid only in the *Active State*.

**retVal moveRelative (in string pointName, in ReferenceFrame frame, in sequence double values)**

Performs a relative move with the device.

<pointName> a string that uniquely identifies the path.

<frame> enumerated keyword (s) describing the type of move: JOINT\_FRAME data is a list of values which specify relative destination in terms of joint values)  
WORLD\_CARTESIAN\_FRAME data is a list of values which specify relative destination in terms of world-based Cartesian coordinates.

BASE\_CARTESIAN\_FRAME data is a list of values which specify relative destination in terms of robot-based Cartesian coordinates TOOL\_FRAME (data is x,y,z,roll,pitch,yaw which specify a location relative to the tool frame).

<sequence> desired move converted to double if the function returns the return code CMD\_STARTED\_OK, then the supervisor should expect an event which tells the supervisor that it has completed or an alarm which indicates an error. If it returns CMD\_EXECUTED\_OK then the function completed and no event monitoring needs to be performed.

**retVal moveAbsolute (in string pointName, in ReferenceFrame frame, in sequence double values)**

Perform an absolute move with the device.

<pointName> string which names the point the robot is going towards. This name will be included in the event data.

<frame> enumerated keyword (s) describing the type of data: JOINT\_FRAME data is a list of values which specify absolute destination in terms of joint values).

WORLD\_CARTESIAN\_FRAME data is a list of values which specify absolute destination in terms of world-based Cartesian coordinates.

BASE\_CARTESIAN\_FRAME data is a list of values which specify absolute destination in terms of robot-based Cartesian coordinates.

<j1>, <j2>, <j3>, <j4>, <j5>, <j6> ... joint values converted to double.

<speed> converted to double and is optional (specifies speed for this move only - always interpreted as a percent of max speed).

Return Code: If the function returns the return code CMD\_STARTED\_OK, then the supervisor should expect an event which tells the supervisor that it has completed or an alarm which indicates an error. If it returns CMD\_EXECUTED\_OK then the function completed and no event monitoring needs to be performed.

**retVal moveArc (in string pointName, in ReferenceFrame frame, in OffsetType offset\_type, in sequence double arcValue, in sequence double goalValue)**

Perform a circular move with the robot to goalValue while arcing in a straight line through arcValue.

<pointName> string which names the point the robot is going towards. This name will be included in the event data.

<frame> enumerated keyword (s) describing the type of data: JOINT\_FRAME data is a list of values which specify absolute destination in terms of joint values).

WORLD\_CARTESIAN\_FRAME data is a list of values which specify absolute destination in terms of world-based Cartesian coordinates.

BASE\_CARTESIAN\_FRAME data is a list of values which specify absolute destination in terms of robot-based Cartesian coordinates.

<offset\_type> one of the following enumerated keywords to designate the type of motion embodied in the downloaded data: ABSOLUTE\_DATA (position data is absolute destination points) RELATIVE\_DATA (position data is relative offsets).

<arcValue> sequence of data representing the arc-through point.

<goalValue> sequence of data representing the arc-through point.

Return Code: If the function returns the return code CMD\_STARTED\_OK, then the supervisor should expect an event which tells the supervisor that it has completed or an alarm which indicates an error. If it returns CMD\_EXECUTED\_OK then the function completed and no event monitoring needs to be performed.

### 3.5 Path Motion Commands

This section presents commands that perform path-defined motions.

#### 3.5.1 Blocking and Locked or Resting States

These blocking commands are valid in both the *Locked and Resting States*.

**retVal pathLoad (in string pathName, in ReferenceFrame frame, in MotionType motion\_type, in OffsetType offset\_type, in int number\_of\_points, in sequence tag\_struct tags)**

Download a path segment which is appended to the current designated motion queue, *three or four parameters*.

<pathName> optional ASCII string which provides an ID for the segment being downloaded (this will be ignored for now, but later the capability will be provided to select and execute paths by name).

<frame> one of the following enumerated keywords to designate which reference frame the data is in WORLD\_CARTESIAN\_FRAME  
BASE\_CARTESIAN\_FRAME JOINT\_FRAME.

<motion\_type> one of the following enumerated keywords to designate the type of motion embodied in the downloaded data: STRAIGHT\_MOTION  
JOINT\_MOTION.

<offset\_type> one of the following enumerated keywords to designate the type of motion embodied in the downloaded data: ABSOLUTE\_DATA (position data is absolute destination points) RELATIVE\_DATA (position data is relative offsets).

<number\_of\_points> integer value indicating the number of points for the segment being downloaded.

<tag\_structs> tag structures containing path data. The tag structure is shown below:

```
typedef struct _tag_struct
{
    double x,y,z,roll,pitch,yaw;
    double jointvals[24];
    string tagName;
    short tagIndex;
    double timeDelay;
    double transitionSpeed;
    double speed;
    double accel;
    sequence string auxCommands;
}
```

Note: Each pathName will be maintained as a separate motion queue. Queues will be maintained simultaneously. For each path name, the first LoadPath command executed will establish a new path when the data is downloaded. Subsequent LoadPath commands will append data to this path until either the path name is cleared with a ClearPath command or an error condition occurs. When LoadPath is executed, a series of data message exchanges occurs between the supervisor and subsystem in order to download the path data.

Each path point will contain 3 major segments of information: the position data, speed control parameters, and auxiliary data. The position data will consist of 6 values (x, y, z, roll, pitch, yaw) if a Cartesian path segment is being downloaded. Otherwise it will contain one joint value for each degree of freedom. If ABSOLUTE\_DATA is designated, each point should be an absolute destination, and if RELATIVE\_DATA is specified, each point should consist of relative offsets. The units will be interpreted according to what was established by the last SetUnits command executed prior to this command. There are 3 speed control parameters, a transition speed (percent of max), relative speed (percent of max), and delay (time in seconds). The auxiliary data will be defined later.

**retVal pathClearPath (in string PathName)>**

Clear the selected motion queue, *one parameter*.

<PathName> Name of path to clear This function will delete the path  
<PathName> from the list of maintained motion queues.

### 3.5.2 Non-Blocking and Active State

These non-blocking commands are valid only in the *Active State*.

#### retVal pathMove (in string pathName, in int number\_of\_cycles)

perform a path move in which a motion queue is built up using the LoadPath command, and then MoveAlongPath is executed.

The system will execute any digital IO or other commands specified in the auxiliary data.

<PathName> name of path to execute.

<number\_of\_cycles> integer value applicable only when the actual path move is being performed; it indicates how many times to repeat the given path move (a 0 implies dry run without aux data activation; if omitted, 1 is assumed).

If the function returns the return code CMD\_STARTED\_OK, then the supervisor should expect an event which tells the supervisor that it has completed or an alarm which indicates an error. If it returns CMD\_EXECUTED\_OK then the function completed and no event monitoring needs to be performed. As the function proceeds along a path, an event will be generated when the robot arrives at each individual tagpoint. The event will have the name of the tagpoint as part of its data.

## 3.6 Asynchronous Interruption Commands

This section presents commands that perform asynchronous interruption commands.

### 3.6.1 Blocking and Moving State

These blocking commands are valid only in the *Moving State*.

#### retVal stopMotion ()

Stop current motion gracefully by generating a deceleration profile which follows the original pre-planned path, *no parameters*.

### 3.6.2 Blocking and Paused or Locked State

These blocking commands are valid only in the *Paused or Locked State*.

**retVal resumeMotion ()**

resume current motion gracefully by generating an acceleration profile which follows the original pre-planned path, no parameters.

**retVal abortMotion ()**

aborts current motion stopped by stopMotion (), and clear the motion queue, no parameters.

**3.7 Sensor Dependent Commands**

This section presents commands that perform Sensor Dependent Commands.

**3.7.1 Blocking Command**

This blocking command is *State Independent*.

**retVal getSensorReading (in string sensor\_type, out sequence double current\_data)**

<sensor\_type> ASCII string which is associated with the name of a particular sensor type, such as: “force\_sensor”, “laser\_standoff\_sensor”, etc.

<current\_data> is the sequence of double data which is associated with the current sensor reading.

**3.7.2 Non-Blocking and Active State****retVal moveReact (in string SensorType, in double MotionOffsets[6], in double MotionVelocity, in double ComplianceVelocity, in double Thresholds[6])**

This non-blocking command is valid only in the *Resting State*.

It will perform a relative move until an activated sensor detects a threshold or event of some kind, *eight parameters + react parameters - thresholds, gains, sensor envelope*.

<x>, <y>, <z> values converted to double specifying maximum distance and direction to move relative to a tool or end effector frame defined at the tool tip by a previous SetToolLength command (will move this far if a sensor threshold is never reached).

<c\_x>, <c\_y>, <c\_z> values converted to double specifying maximum compliance distance and direction to move which should be orthogonal to the regular motion vector just defined (this normally will be a zero vector for MoveReact, meaning no compliant motion, but rather just a “mov-til-touch” motion).

<speed> converted to double (specifies speed for this move only - always interpreted as a percent of max speed).

<c\_speed> converted to double (specifies compliance speed for this move only - should be 0 for MoveReact).

<t1>, <t2>, <t3> thresholds for the active sensor (currently only one non-zero threshold should be specified, and if a sensor reading for this component exceeds it, motion stops).

<g1>, <g2>, <g3> gains which influence the responsiveness of the algorithm (currently only the first gain is utilized by the algorithm, the others are ignored).

<min1>, <min2>, <min3> define the lower bounds for an envelope placed around the sensor data to eliminate spikes (currently ignored).

<max1>, <max2>, <max3> define the upper bounds for an envelope placed around the sensor data to eliminate spikes (currently only the first value is used, and if a sensor reading exceeds this value, motion is terminated).

Note: These arguments may change or be interpreted differently in later releases based on our experience gained from testing various types of react motion using different sensors and different manipulators. If the function returns the return code CMD\_STARTED\_OK, then the supervisor should expect an event which tells the supervisor that it has completed or an alarm which indicates an error. If it returns CMD\_EXECUTED\_OK then the function completed and no event monitoring needs to be performed.

**retVal comply (in string SensorType, in double Thresholds[6], in double MaxTimeout, in short PanicStopCode, in double MaxTravelDistances[6]), in double ProportionalGains[6]), in double DifferentialGains[6])**

This non-blocking command is valid in both the *Resting and Moving States*.

It will perform a relative move while maintaining a continuous sensor reading in the direction normal to motion; sensor readings are used to compute perturbations which are added to the originally specified move in order to maintain compliance; *eight parameters + compliance parameters - thresholds, gains, sensor envelope*:

<x>, <y>, <z> values converted to double specifying maximum distance and direction to move relative to a tool or end effector frame defined at the tool tip by a previous SetToolLength command.

<c\_x>, <c\_y>, <c\_z> values converted to double specifying maximum compliance distance and direction to move which should be orthogonal to the regular motion vector just defined.

<speed> converted to double (specifies speed for this move only - always interpreted as a percent of max speed).

<c\_speed> converted to double (specifies compliance speed for this call).

<min1>, <min2>, <min3> define the lower bounds for an envelope placed around the sensor data to eliminate spikes (currently ignored).

<max1>, <max2>, <max3> define the upper bounds for an envelope placed around the sensor data to eliminate spikes (currently only the first value is used, and if a sensor reading exceeds this value, motion is terminated).

Note: These arguments may change or be interpreted differently in later releases based on our experience gained from testing various types of compliant motion using different sensors and different manipulators. If the function returns the return code CMD\_STARTED\_OK, then the supervisor should expect an event which tells the supervisor that it has completed or an alarm which indicates an error. If it returns CMD\_EXECUTED\_OK then the function completed and no event monitoring needs to be performed.

### 3.7.3 Blocking and Locked State

These blocking commands are valid only in the *Locked State*.

#### retVal manualControl (in string type)

control the robot with some manual input device, *one parameter*.

<type> ASCII keyword specifying the type of manual control desired: teachpendant  
spaceball off.

## 3.8 EndEffector Commands

This section presents commands that perform EndEffector Commands.

### 3.8.1 Blocking and State Independent

These blocking commands are state independent.

#### retVal getEndEffectorState (out string currentEE, in string EEState,.....,)

<currentEE,> the name of the tool.

<EEState> string which describes the state of the tool.

#### retVal getTool (in string tool\_ID)

pick up a tool with the robot, *one parameter*:

<tool\_ID>ASCII string identifying the desired tool

**retVal putTool (in string tool\_ID)**

put away a tool with the robot, *one parameter*:

<tool\_ID>ASCII string identifying the desired tool

**retVal activateTool (in string tool\_ID, in long BitField)**

**retVal deactivateTool (in string tool\_ID, in long BitField)**

## 4.0 DIGITAL\_IO IDL

This section describes the DIGITAL\_IO IDL command set. The digital I/O interface will work as follows: The set function will allow a user to set I/Os, the get function will allow the user to query the I/Os, and the signalDigitalIO will allow the user to set up a callback on the IO, which will get invoked when the specified bits have been set.

### 4.1 Blocking and State Independent

These blocking commands are state independent.

**retVal setDigitalIO (in string type, in short unitID, in long bitsON, in long bitsOFF, in string description)**

<type> string which describes the type of IO being performed.

<unitID> short which describes which IO to toggle.

<bitsON> long which is the BitField of I/Os to activate.

<bitsOFF> long which is the BitField of I/Os to deactivate.

<description> string which describes the operation.

**retVal getDigitalIO (in string type, in short unitID, out long bitsON, in string description)**

<type> string which describes the type of IO being performed.

<unitID> short which describes which IO to query.

<bitsON> long which is the BitField of I/Os which are active.

<description> string which describes the operation.

**retVal signalDigitalIO (in string type, in short unitID, in long bitString, in short pollingTimeOut, in short pollingCycleTime, short callback\_type)**

<type> string which describes the type of IO being performed.

<unitID> short which describes which IO unit to monitor.

<bitString> long which is the BitField of I/Os to which are being monitored for change to a set state.

<callbackType> flag which specifies which function calls the client.

**retVal sendCommand (in string input, out string result)**

Vendor hook for custom commands.

```
typedef struct _retval
{
    short ret_code;
    string ret_msg;
} retVal;
```

## 5.0 Return Codes

This section describes the Return Codes. For coding convention, return codes greater than or equal to 0 denote success and return codes less than 0 denote errors.

### 0 CMD\_EXEC\_OK

This return code means that for blocking commands the command was executed and completed with no errors.

### 1 CMD\_STARTED\_OK

For non-blocking commands this means the command was understood and started execution successfully.

### -1 CMD\_ERR

Generic error return code.

### -2 CMD\_PARAM\_ERR

Error parsing one of the parameters.

**-3 CMD\_PARAM\_OUT\_OF\_RANGE\_ERR**

Parameter is out of range.

## **6.0 Parameter Constants**

This section describes the LandScape Parameter Constants.

**enum OffsetType { ABSOLUTE\_DATA, RELATIVE\_DATA};**

**enum ReferenceFrame { JOINT\_FRAME, BASE\_CARTESIAN\_FRAME,  
WORLD\_CARTESIAN\_FRAME, TOOL\_FRAME};**

**enum MotionType { JOINT\_MOTION, STRAIGHT\_MOTION, SLEW\_MOTION,  
CIRCULAR\_MOTION};**

**enum SpeedType { PERCENT\_SPEED, LINEAR\_SPEED, ANGULAR\_SPEED};**

**enum AccelType { PERCENT\_ACCEL, LINEAR\_ACCEL, ANGULAR\_ACCEL};**

Distribution:

- 5 MS1004 ISRC Library, 9623
- 1 MS1004 Michael McDonald, 9623
- 1 MS1006 Ross Burchard, 9671
- 1 MS1010 Dan Small, 9622
- 1 MS9018 Central Technical Files, 8940-2
- 2 MS0899 Technical Library, 4916
- 1 MS0619 Review and Approval Desk, 15102  
for DOE/OSTI