NISTIR 5658

Unified Telerobotic Architecture Project (UTAP) Standard Interface Environment (SIE)

Ronald Lumia*
John Michaloski
Robert Russell
Thomas Wheatley
Intelligent Systems Division

and

Paul Backes
Sukhan Lee
Robert Steele
Jet Propulsion Laboratory
NASA
Pasadena, CA 91109

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Bldg. 220 Rm. B124
Gaithersburg, MD 20899

NIST

*Ronald Lumia is currently working at the University of New Mexico
in the Mechanical Engineering Department. His address is:
The University of New Mexico
Albuquerque, NM 87131

NISTIR 5658

Unified Telerobotic Architecture Project (UTAP) Standard Interface Environment (SIE)

Ronald Lumia*
John Michaloski
Robert Russell
Thomas Wheatley
Intelligent Systems Division

and

Paul Backes
Sukhan Lee
Robert Steele
Jet Propulsion Laboratory
NASA
Pasadena, CA 91109

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Bldg. 220 Rm. B124
Gaithersburg, MD 20899

May 1995

Contents

Figures

Foreword

Under the sponsorship of the Air Force Material Command (AFMC) Robotics and Automation Center of Excellence (RACE) at Kelly Air Force Base, San Antonio, TX, the Unified Telerobotic Architecture Project was funded to define an open architecture to improve the efficiency and productivity of the maintenance operations. The UTAP specifies an open architecture for telerobotics along with specific implementation options designed to assist the work at Air Force maintenance facilities.

The status of the UTAP Standard Interface Environment - SIE - has progressed to the point that the architecture remains stable and the project has seen the interfaces evolve significantly after several Design Reviews. To date, the emphasis of review effort has been on the remote interfaces containing the real-time control elements. Additional work remains to validate the functionality of the interfaces, resolve configuration and integration issues, solidify the interface environment and substantiate the validation and conformance process.

Disclaimer

Electronic Access to Document

A compressed copy of this document in Postscript format and the related source code in shar format is available electronically. Access to the UTAP report and UTAP source code is available through the Internet standard File Transfer Protocol (ftp) . The ftp site name is "giskard.cme.nist.gov". Directions for an ftp session to retrieve the report and source code follow.

First, change directory to your local destination directory. Next use the command "ftp" to remotely login using "anonymous" for the name, and give your email address for the password. This will allow you reading and copying privileges.

```
you[1]% cd your_local_directory
you[2]% ftp giskard.cme.nist.gov
Connected to giskard.cme.nist.gov.
220 giskard FTP server (Version wu-2.3(1) Wed Apr 6 14:21:22 EDT 1994) ready.
Name (giskard.cme.nist.gov:yourname): anonymous
331 Guest login ok, send your complete e-mail address as password.
Password: "your complete e-mail address"
230-      Welcome to the FTP server for the Intelligent Systems Division
230-          National Institute of Standards and Technology
230-                          Gaithersburg, MD
230-
230-Please read the file README
230-  it was last modified on Mon Dec  6 11:57:23 1993 - 136 days ago
```

```
230 Guest login ok, access restrictions apply.
```

Once connected, change into the "utap" directory containing the desired files.

```
ftp> cd pub/utap
250 CWD command successful.
```

To get everything at once, a compressed tar version of the documentation and a shar bundle of the source files is in the main directory. ALWAYS USE BINARY/IMAGE MODE TO TRANSFER THESE FILES! Text mode does not work for tar files or compressed files.

```
ftp> mget utap.doc.tar.Z utap.src.shar
```

Terminate the ftp session with the quit command.

```
ftp> quit
```

Assuming a UNIX environment, one will be required to unbundle the files. For the documentation, uncompress and extract the documentation files with tar. The source is in shar format, so use unshar to unbundle files.

```
you[1]% (mkdir doc; cp utap.doc.Z  doc; cd doc; \
        uncompress utap.doc.tar.Z; tar -xf utap.doc.tar; )

you[2]% (mkdir src; cp utap.src.shar  src; cd src; \
        unshar utap.src.shar;)
```

The documentation is in Postscript format (*.ps).

Introduction

This introduction is not considered part of the proposed standard.

The purpose of this Working Draft Standard Document is to define a common architecture for telerobotics systems for use in Air Force applications with great dual-use potential for civilian applications. There are various Air Force applications, such as paint stripping and painting, surface finishing, and skin cutting which can benefit from the integration of telerobotics systems tools.

Telerobotics systems will enable human operators, who now execute these tasks manually, to operate telerobots to execute the tasks faster, safer, and with higher quality. Telerobotics aims at the integration and fusion of the strengths of machine and human to extend the capabilities of either. Telerobotics transcends the human barriers of space, time, power, speed, accuracy, and safety as well as the machine barriers of cognition, understanding, reasoning, and planning. Besides the conventional applications of telerobotics technology to space, underwater, nuclear, and mining operations, telerobotics technology may be applied to the semi-automation of industrial processes where the full robotic automation is difficult to implement but the manual operation is too costly to practice. The Air Force application domains of aircraft maintenance and remanufacturing are good candidates for successful telerobotics insertion due to their small batch sizes, partially modeled task environment, and physically challenging and hazardous work environments. In aircraft maintenance and remanufacturing applications the operator may provide the high level cognitive planning and sensory perception, which are currently difficult to provide in a robotic system, while the robot provides precise control and works in dangerous environments.

It is intended that commercial telerobotics applications will become feasible due to the specification of an architecture and standardizing the components of the systems. This will allow systems to be built from standard hardware and software modules which, rather than being custom developed, can be reused from other applications or purchased. The architecture therefore provides a framework for design and implementation of telerobotics systems for different telerobotics applications while utilizing a common architecture and hardware and software modules. The customization in developing a system will be in the selection of which modules to use rather than in development of all the modules. This will allow both minimal, i.e., inexpensive, and complex, i.e., expensive systems to be built using the same architecture.

Maintenance of systems developed with standard modules will likely be less expensive and cause less system down time than for custom systems. Service personnel will be easier to find since their skills will have wider applicability than those of people who are familiar with a custom system.

An important feature of the architecture from the operator's viewpoint is a common operator interface across different applications. The various application interfaces will be easier to learn, remember, and use. This will reduce training time and costs, as well as provide more skillful and reliable operators.

Audience

The intended audience that this Working Draft Standard Document has been developed for:

a) Air Force Maintenance System Designers

b) Control System Designers/Engineers

c) Control System Integrators

d) Telerobotic Control Applications Programmers

e) Hardware and Software Purchases

f) End-users operating a `SIE` controller

Organization of the Working Draft Document

The Working Draft Standard Document is divided into 9 parts.

- Scope
- References
- Definitions and global concepts
- Conformance
- Architecture Reference Model

- Interface Environment
- Information Models
- Configuration and Integration
- Interface Framework

A series of annexes follow the Working Draft Standard Documentthat contain normative and informative reference material.

Background

The UTAP architecture definition utilized telerobotics research and development results from universities and national laboratories, previous studies, and current robotics off-the-shelf capabilities. Most of the required capabilities have been demonstrated in prototype systems, but without a common architecture approach. The unified architecture specifies the hardware and software modules so that telerobotics systems can be built from standard commercial components.

The architecture described in this report is a refinement of the architecture described in an earlier study [JPL]. That study provided a high level description of the unified architecture and its components. The unified architecture is an integration of many open architecture technologies.

At NIST, the unifying architecture for system development has been the Real-time Control System (RCS) [RCS] that has evolved from cerebral models of brain behavior into a general theory of intelligence. In addition to the RCS architecture, a methodology accompanies the architecture for the analysis, design and implementation of control systems. The importance of the RCS lies in the abstractions and generalizations it forwards in pursuit of open solutions that apply beyond the demands of any one application.

Another related architecture is the architecture associated with the Next Generation Controller (NGC) project [SOSAS]. It is intended that this UTAP architecture be an Application Architecture for an NGC system. The modules of the system are therefore described as components of an NGC system with specified responsibilities and interfaces. A specific NGC profile is not specified since that would be selected for a specific Application System.

Purpose

Several principles guided the development of this Working Draft Standard Document.

Open Architecture Technology:
Openness provides benefits and savings through flexibility and extensibility but does not address portability. Interfaces under one vendor's open architecture generally will not run under another vendor's system. Openness is the first step towards standardization. Requirements for a standard "open solution" include the ability to allow the development of controllers by users or system integrators who want to piece together their own systems component by component, modify the way their controller does certain things, apply their modifications to

another controller, or start small and upgrade as they grow. These basic open architecture requirements include:

Modularity: Refers to the ability of controls users and system integrators to purchase and replace components of the controller without unduly affecting the rest of the controller.

Extensibility: Refers to the ability of intelligent users and third parties to incrementally add functionality to a module without replacing it completely.

Portability: Refers to the ease with which a module can run across platforms Standards such as ANSI C and POSIX are required to serve as a reference to which programmers adhere.

Scalability: Like portability, refers to the ease with which a module can be made to run in a controller based on another platform, but unlike portability, scalability allows different performance based on the platform selection. Scalability means that a controller may be implemented as easily by systems integrators on a high-speed processor, as a distributed multi-processor system, or on a standalone PC.

Applying Today's Technology:
The UTAP is intended as a `SIE` for immediate use. One could overestimate the real-potential of systems in developing the scenarios, and become mired in the range of possibilities and expectations of an architecture. For the UTAP architecture and interfaces, it is assumed that a reasonable level of effort and Commercial Off The Shelf (COTS) equipment are immediately available and can be used to solve the applications tasks.

Another assumption was that innovation would be minimized. Innovation affects both the "how-to" and "what is in" when defining interfaces. For the how-to, should the interfaces use established, but sometimes flawed, approaches, or should the interface adapt newer but evolving and unproven approaches? There are established efforts for interface definitions that are very elegant (e.g., [STEP], [CORBA]), but are either not cost-effective or still suffering growing pains. The UTAP will start with a baseline of a simple strategy and concentrate on the "what is in" the interfaces instead of dwelling on the "how-to" pass information within an interface. It will be assumed that at some point an industry-standard for manufacturing-based application interface communication and infrastructure will have evolved.

Another question is the amount of scientific pioneering of new technology expected within the interfaces, e.g., "What are the functions that should be incorporated into the next generation of commercially available sensors?" For instance, should an array sensor - such as a range sensor - return curvature identification? For the UTAP interfaces, we did not attempt to innovate new definitions, but rather, attempted to standardize on established technology. However, the level of innovation within an interface is not compromising since the UTAP modules and interfaces are scalable.

Focus on Interface Content, Not Interface Transport:
An interface has two critical issues. One issue is the method, or "How-to-pass," which describes how one will represent the language and the perform the communication. The other issue is interface knowledge or the "What-to-pass" within the interface. The interface knowledge is tied to the application requirements and must match the needs for the command, control, status and synchronization of the system. The "How-to-pass" issue is guaranteed to wrap one around the axle. Obviously, one cannot be blind to the how-to-pass elements of the interface - protocols, configuration and language style greatly impact the Interface Framework.

Test Validation

The first validating implementation of the architecture will be done using a commercial controller. The hierarchy of components associated with major commercial robotics systems for robot control is shown in figure 1. Higher level components are supported by lower level com-

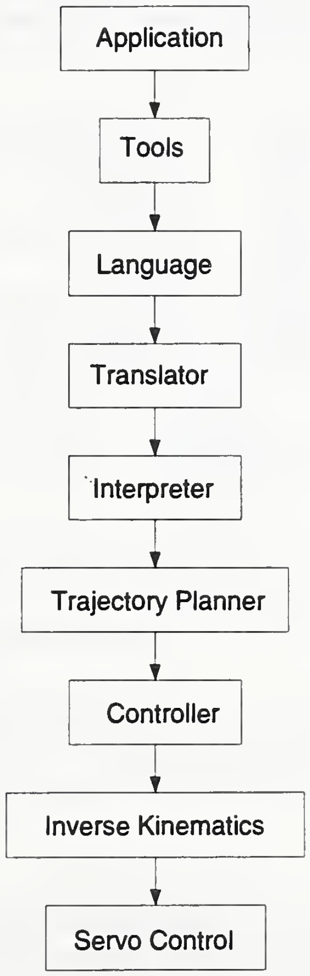

Figure 1 – Commercial robotics components hierarchy

ponents. The Servo Control component provides servo control of the joint angles. It has joint angle commands for inputs. The Inverse Kinematics component transforms task level commands into joint commands, e.g., pose of the tool into joint angles which result in that tool pose. The Controller provides the task level control including merging Cartesian trajectories with task level sensor based control. The Trajectory Planner generates the planned trajectory, e.g., using a trajectory generator to generate Cartesian setpoints for the tool to pass through. The Interpreter interprets and sequences the task program commands. The Translator translates language source commands into intermediate p-code commands which are more efficient to execute than the language commands. The Language is a general purpose robotics task description language which provides all capability needed to support the desired applications. The Tools are software packages specific to application domains which provide macro commands which can be used to efficiently develop application programs. The Tools may also provide an environment for developing application programs. The Application is the application program for a specific application. It will be developed with commands from the Tools package(s) and the supporting language. Ideally, the application program will be developed using only the Tools packages and Tools supporting development environment. Another way to envision this hierarchy of components is by combining components into components which are commonly separate parts of a robotic system, as shown in figure 2. The controller is the robot controller, e.g., Fanuc, Adept, or

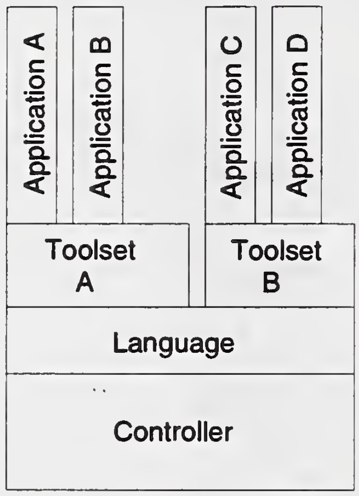

Figure 2 – Robot system hierarchy

Trellis. The language is the robot language, e.g., Karel, SIL or V+. The Tools and Applications are the same as the Tools and Applications above.

Conformance

In publishing this Working Draft Standard Document, the Working Group intends to provide a yardstick against which various control implementations can be measured for conformance. It is not the intent of the Working Group to measure or rate any products, or reward or sanction any vendors of products for conformance or lack of conformance to this standard, nor will any attempt to enforce this standard by these or any means.

It will be assumed that individuals who are evaluating the product will be able to attach and run a *test and verification* harness for a particular module. An entire controller would be tested and verified for conformance through the process of harness rewiring to accept one, two, ... n modules.

Extensions

Activities to extend this Working Draft Standard Documentfor additional requirements are anticipated. This is an overview of how extensions to the standard will be done and how users of the standard can keep track of that status.

Extensions are provided as *Supplements* to this document. Supplements may contain either required functions or optional facilities. Supplements may add additional conformance requirements defining new classes of conforming systems or applications.

Supplements are not used to provide a general update of the standard. Standard revisions are done through the review procedure as specified by the standard body. Supplements currently under consideration at this time include:

- CORBA Interface
- IDL or ASN.1 Interface Definitions

Typographic Conventions

This Working Draft Standard Document uses the following typographic conventions:

a) The *italic* font is used for the initial appearance of defined terms; and cross references to defined terms within the definitions terminology.

b) The **bold** font is used for C and C++ language types; references to other sections or chapters.

c) The `constant-width` font is used to illustrate examples of code.

Related Standard Work

This Working Draft Standard Document was prepared by a Working Group under the leadership of the RACE with the intention to standardize this effort within a Technical Standards Committee. At the time this working draft was distributed, the membership of the Working Group was as follows:

Working Group

Paul G. Backes	John L. Michaloski	Robert Steele
Michael Leahy	Scott Petrosky	Albert Wavering
Sukan Lee	Francois Pin	Thomas Wheatley
Ronald Lumia		

Unified Telerobotic Architecture Project — Standard Interface Environment — Working Draft Document

1 Scope

This Working Draft Standard Documentis intended to serve as a guide in the system design and implementation of telerobotic systems, to minimize the variety of system interfaces and to promote a unified approach to building telerobotic systems and to foster the interchangeability of telebrobotic architecture components. It is intended to provide scalable complexity to accommodate simple systems and at the same time be systematically extensible to accommodate more complex systems.

The standard presents a reference model architecture and SIE for telerobotic applications. The standard contains general-purpose concepts and presents terminology definitions for the architecture and the interface between components.

2 References

[JPL] NASA JPL, "A Generic Telerobotics Architecture for C-5 Industrial Processes," Final Report Prepared for Air Force Material Command (AFMC), Robotics and Automation Center of Excellence (RACE), San Antonio Air Logistics Center, Kelly AFT, TX 78241.

[ASN.1a] Information Processing - Open Systems Interconnection - Abstract Syntax Notation One (ASN.1); International Organization for Standardization and International Electrotechnical Committee, 1987, International Standard 8824.

[ASN.1b] Information Processing - Open Systems Interconnection - Abstract Syntax Notation One (ASN.1) - Draft Addendum 1: Extensions to ASN.1; International Organization for Standardization and International Electrotechnical Committee, 1987, International Standard 8824/DAD 1.

[CORBA] Object Management Group. Object Management Architecture Guide, Document 92.11.1, Framingham, MA, 1991.

[EIA274] "EIA Standard - EIA-274-D, Interchangeable Variable, Block Data Format for Positioning, Contouring, and Contouring/Positioning Numerically Controlled Machines," Engineering Industries Association, Washington, D.C., February 1979.

[EIA441] "EIA Standard - EIA-441, Operator Interface Functions of Numerical Controls," Engineering Industries Association, Washington, D.C., January 1979.

[MMS1] ANSI/EIA-511 part 1, 1989 - Manufacturing Message Specification (MMS) - Service Definition.

[MMS2] ANSI/EIA-511 part 2, 1989 - Manufacturing Message Specification (MMS) - Protocol Definition.

[MMS1924] ANSI/EIA Standard Proposal No. 1924 - A Proposed New Companion Standard to EIA-511, "Numerical Control Message Specification," (if approved, to be published as ANS1-19506-4/EIA-566).

[OSI] "Open Systems Interconnection: definition of common application service elements," International Standards Organization.

[POSIX] "POSIX (Portable Operating System Interface), ANSI/IEEE Std 1003.1-1988" or FIPS-PUB-151-1.

[RS441] "EIA Standard 441, Operator Interface Functions of Numerical Controls," Electronics Industries Association, Washington, D.C., January 1979 (Reaffirmed.July 14, 1992).

[RS274D] "EIA Standard - EIA-274-D, Interchangeable Variable, Block Data Format for Positioning, Contouring, and Contouring/Positioning Numerically Controlled Machines," Engineering Industries Association, Washington, D.C., February, 1979.

[STEP41] "ISO 10303-41 Industrial Automation Systems and Integration Product Data Representation and Exchange - Part 41: Integrated Resources: Fundamentals of Product Description and Support."

[STEP42] "ISO 10303-42 Industrial Automation Systems and Integration Product Data Representation and Exchange - Part 42: Integrated Resources: Geometric and Topological Representation."

[SOSAS] National Center for Manufacturing Sciences, "Next Generation (NGC) Specification for an Open System Architecture Standard (SOSAS), Revision 2.5", August 1994.

3 Definitions

3.1 Standards Terminology

3.1.1 defined: A value or behavior is *defined* if the implementation defines and documents the requirements for correct program construct and correct data.

3.1.2 may: With respect to conformance, the word *may* is to be interpreted as an optional feature that is not required in this standard but can be provided.

3.1.3 shall: With respect to conformance, the word *shall* is to interpreted as a requirement on the implementation for strict conformance.

3.1.4 should: With respect to conformance, the word *should* is to interpreted as not a strict requirement, but interpreted as a necessary courtesy for explaining non-standard additions and extensions.

3.1.5 supported: Certain functionality in this standard is optional, but the interfaces to that functionality are always required. If the functionality is *supported*, the interfaces work as specified by this standard (except that they do not return the error condition indicated for not-supported case). If the functionality is *not supported*, the interface shall always return the indication specified for this situation.

3.1.6 undefined: A value or behavior is *undefined* if the standard imposes no portability and interoperability requirements on applications for erroneous program construct, erroneous data, or use of an indeterminate value. Implementations (or other standards) may specify the result of using that value or causing that behavior.

3.1.7 unspecified: A value or behavior is *unspecified* if the standard imposes no portability requirements on applications for correct program construct , correct program data, or correct program interoperability.

3.2 General Terms

3.2.8 API: The term API refers to a type of interface in which one has a data representation and set of functions associated with the data representation. By contrast for example, Postscript is an interface *language* for printers. For an API, the data and function abstraction (in Smalltalk OO lingo, class and methods) hides the underlying physical representation or implementation from the programmer. As an example, C is a general-purpose language (CPU interface) which contains many application-specific API libraries, such as math, or a socket library as an API abstraction for TCP/IP communication. For the math library, one has a representation of the data (a double in IEEE floating point, a set of functions (e.g., sin, cos, atan, etc.) which hide whether the computation is done on FPU hardware or in software.

The environment is important in specifying options. Through the use of compiler switches one can specify an platform environment for a FPU or not.

3.2.9 build: An open-architecture controller is built from modules and component parts. The operation to build a controller from module components is multi-faceted and includes the following:

– User defines "initial conditions" such as hardware, peripherals, (i.e., computing resources in general)

– Platform supplies system low-level services (e.g., file-mgmt, etc.)

– Integrator wires selected modules together

– Modules need to support user-specification of timing requirements

– Supply of "dummy" or minimal modules where user has not selected any

– Desirability to have convenient ways to experiment: reconfigure modules quickly and (not required) capture their results in order to organize your experimentation

3.2.10 channel: A *channel* (or transport) is the abstract connection between communicating modules along which the message is transferred, e.g. network, shared memory, local procedure call, remote procedure call, software interrupt, event, signal, network, stream, mailbox, etc.

3.2.11 connection: A connection requires two (or more) processes to communicate via a connection. One module is the *sender* (or writer) and one (or more) module is the *receiver* (or reader). A good analogy to this paradigm is a telephone conversation. When you initiate a telephone call, you are initiating a connection. The other party hears the telephone ring, and then answers the phone to complete the connection. How the connection is actually made is the responsibility of the lower-layer service (the telephone companies handle the underlying hardware and communication protocol). The conversation consisting of an agreed upon language and dialogue protocol is equivalent to the application session layer or Open System Interconnection (OSI) [OSI] layer 7.

3.2.12 component: A component definition will adopt the NGC SOSAS [SOSAS] concept of a *reference architecture* consisting of *primitive* and *aggregate components. Components* are defined as abstract building block elements that describe functionality and communication. The *application architecture* is built from these components. Components have the following attributes:

– responsibility;

– peer-to-peer or collaborative relationships;

– behavior (specific functionality encapsulated by the component);

– messages, that is, the complete set of specific instructions necessary for invoking all of the behaviors encapsulated by the component;

– Application Program Interface(s) or the interfaces a component uses specifically to access services provided by the SOSAS notion of an Open Systems Environment.

3.2.13 data encapsulation: API is a part of the notion of data and *functional* encapsulation and the concept of data hiding. Data encapsulation refers to the object-oriented idea of grouping the data and functions into a class container (or black box.) Thus, a queue class specification offers the user a general data representation (e.g., circular list) with a set of functions (create, add, remove, delete, front, ...) bundled under the QUEUE class. More interesting is the notion of abstracting the queue elements (say a queue of integers vs. a queue of floats) allowing a user to specify the element type since the functions are identical (e.g., which could be implemented with an ADA generic or C++ templates.)

3.2.14 interface: An interface is a *connection* between modules. The *interface* is defined by the language the communicating modules use to exchange information. The language is the formal system of signs and symbols and rules for formulation (syntax) and transformation of admissible expressions. For terms of this Working Draft Standard Document, two types of interfaces will be discussed, *programmable interface* and *published interface.* A programmable interface describes messages as programs passed between modules that would explicitly contain data structure declarations, data definition, program flow and actual data. *Published interfaces* describe data size and ordering (or data structure) à priori as the method to specify the syntax of the language.

NOTE 1 – Programmable interface languages contain special keywords or primitives to simplify the process. For example, the Postscript [1)] language contains special-purpose keywords that denote drawing primitives. An appropriate list of primitives is critical to the success of an interface. For a Postscript interface, instead of sending a thousand points to define a shape, one invokes a Postscript primitive shape function with specific parameters. In this case, you send textual "programs" across the interface (e.g., those written in Postscript) instead of raw data. Extending the language with user-defined primitives (e.g., subroutines or macros) is also available within a programmable interface. Within Postscript, one can extend the interface by defining user-shape functions and invoking them with a subroutine calls and a parameter list.

The programmable interface is a powerful, yet costly technique. It requires a high computational overhead to interpret messages. Time is a luxury that cannot be afforded in much of the UTAP architecture. To achieve high performance, many interfaces limit messages to raw data consisting of a keyword and parameter list, formatted according to a published interface definition or Application Programming Interface (API). Such interfaces have a low-overhead and are simple to interpret. The published interface would list acceptable keyword and parameter syntax describing the module functionality and data representation for an interface. Such interfaces can be as simple as a subroutine keyword and parameter list. Distributed interfaces require an additional level of packaging - a sender prepares a message for transport along a channel to the receiver module.

3.2.15 message: A *message* is an instance (or program) written in the interface language. The receiver interprets the message from the sender.

3.2.16 module: A module is a collection of similar computational services. Modules contain software *components* such as C++ classes or ADA packages. A module consists of more than a box of functionality with an explicit Application Programming Interface. Modules consist of:

– A set of functions

– API's for those functions

– A registration process that can be invoked wherein the module registers with the system

[1)]Postscript is a registered trademark of Adobe System, Inc.

being configured what its capabilities are

– An auxiliary store/database containing the specifications for the current instantiation of a module.

3.2.17 open system: IEEE Controls Magazine defines an open system standard as "a specification developed by a consensus process to which any vendor can build products". The following features are characteristics of "openness":

– Products are implemented to internationally agreed standards. Ideally, internationally agreed de jure standards are preferred to de facto standards, but the latter are often used in practice. To be appropriate, a de facto standard must have a large base of independently developed applications available, be supported on a range of different hardware, can be licensed for use by anyone, and have international support.

– Standards are nonexclusive, nonproprietary, and vendor independent. A standard satisfies this requirement if an agreed definition is publicly available, the specification is not owned or controlled by a company or group of companies with vested commercial interests, and no restrictions are imposed on its use.

– Applications can be moved as necessary between systems of different makes and sizes. This is more than a simple matter of application portability. It is also a means of ensuring that data and user experience is also portable between the same application on different hardware systems.

– Usable information can be exchanged when required between different systems. This ensures that data is usable by different applications thereby ensuring that different applications can work together."

3.2.18 protocol: The protocol describes the message passing mechanism and the method in which each module acknowledges receipt of a message, e.g. ack/nack, guaranteed delivery, in-order, blocking/non-blocking, time-out, buffering, queuing, persistent, dynamic. The connection defines the configuration of the interface, e.g., point-to-point, broadcast, blackboard.

3.2.19 telerobotics: Telerobotics methods can be separated into three types: manual control, supervisory control, and fully automatic control. The distinction between these methods is briefly described here. The term *teleoperation* may be used generically to describe all telerobotics methods but is used here in its more common connotation of manual control. In *manual control*, all robot motion is specified by continuous input from a human, with no additional motion caused by a computer. In *supervisory control*, robot motion may be caused by either human inputs or computer generated inputs. In *fully automatic control*, all robot motion is caused by computer generated inputs.

There are two primary subsets of supervisory control: supervised autonomy and shared control. The distinction between them is the nature of the inputs from the operator. In *shared control*, operator commands are sent during execution of a motion and are merged with the closed loop motion generated automatically. Therefore, in shared control, all inputs from the operator are not known à priori to execution of a motion since inputs during execution are also used. In *supervised autonomy*, autonomous commands are generated through human interaction, but sent for autonomous execution. A command can be sent immediately or iteratively saved, simulated, and modified before sending it for execution on the real robot. Also, individual commands can be complete descriptions of the motion or module commands specifying only modifications to the control or monitoring of a specific module of the remote system.

4 Abbreviations

For the purposes of this standard definition, the following abbreviations apply.

ADS	Analysis and Diagnosis Module
API	Application Programming Interface
CORBA	Common Object Request-Broker Architecture
COTS	Commercial Off The Shelf
DCE	Distributed Computing Environment
DB	Data Base
DLL	Dynamically Linked Library
OC	Object Calibration
OI	Operator Input Devices
OM	Object Modeling
OK	Object Knowledge
OSF	Open Software Foundation
POSIX	Portable Operating System Interface for Computer Environments
PTPS	Parent Task Program Sequencer
RSC	Robot/Axis Servo Control
SC	Sensor Control

SGD	Status Graphics and Displays
SIE	Standard Interface Environment
SOSAS	Specification for an Open System Architecture Standard
SS	Subsystem Simulators
TC	Tool Control
TD	Trajectory Description
TDS	Task Description and Supervision
TK	Task Knowledge
TPS	Task Program Sequencer
TLC	Subsystem Task Level Control
TRD	Trajectory Description
VS	Virtual Sensor
XDR	External Data Representation

5 Conformance requirements

5.1 Implementation Conformance

A *conforming implementation shall* meet all of the following criteria:

a) The system *shall* support all required interfaces defined within the standard. These interfaces *shall* support the behavior described herein. The algorithms or other internal mechanisms used to achieve these behaviors is not specified by the standard.

b) The system *may* support additional features or facilities not required by this standard. Nonstandard extensions *should* be identified as such in the documentation. Nonstandard extensions, when used, *may* improve the behavior of functions or facilities defined by this standard, but *shall* maintain basic performance behavior. In the case of nonstandard extensions, the documentation *shall* define an environment in which an application can be run with the behavior specified by the standard. In no case *shall* such environment require modification of a *strictly conforming application.*

5.2 Environment Conformance

A module *shall* conform to the environment as indicated by the configuration file. The environment definition *shall* conform to the *profile* specification as defined in C that complies with the NGC Open System Environment framework [SOSAS]. Other conformance issues remain to be resolved.

5.3 Documentation Conformance

A document with the following information *shall* be available for an implementation claiming conformance to the standard.

This document *shall* contain a conformance statement that indicates the full name, number and date of the standard that applies.This document *shall* contain a conformance section that lists other software standards used to satisfy the infrastructure.

This document *should* specify the behavior of the implementation of the standard where implementation may vary.

This document *should* specify the time-based performance of the implementation of the standard where implementation may vary.

Modules complying with this standard will supply a document that describes the *environment profile* as given the NGC Open System Environment framework [SOSAS] which is defined in Annex C.

6 Application Architecture

The UTAP application architecture is defined so as to avoid point solutions to specific applications. Instead, the UTAP architecture accommodates different types of robotic manipulators with different degrees of freedom, accommodate different part materials and part geometries, new tasks in the workplace, and provide a facility to upgrade/change equipment, sensors, and feedback mechanisms as technology advances.

A *reference model architecture* is a guide as to how to structure the components in a system. Depending on the application, a similar, but not necessarily duplicate instance of the reference architecture may be developed. The goal of the reference model architecture is to model the relationships among *elemental components* that may exist in any system. The goal of the reference model is to provide a framework as to how to organize system components. Figure 3 shows the UTAP application architecture in terms of its elemental components. The architecture includes both implementation and execution features although implementation and execution would be done at different times by different people. Central to the architecture is the application program. This is the program which is run by the operator to execute the telerobotic task. The application program is separated into subsystem task programs and a parent task program. A subsystem is characterized by having a separate task program. There may be separate task programs running on separate controllers for different robots or mechanisms, or separate task programs running on the same controller hardware. Coordinated control between separate task

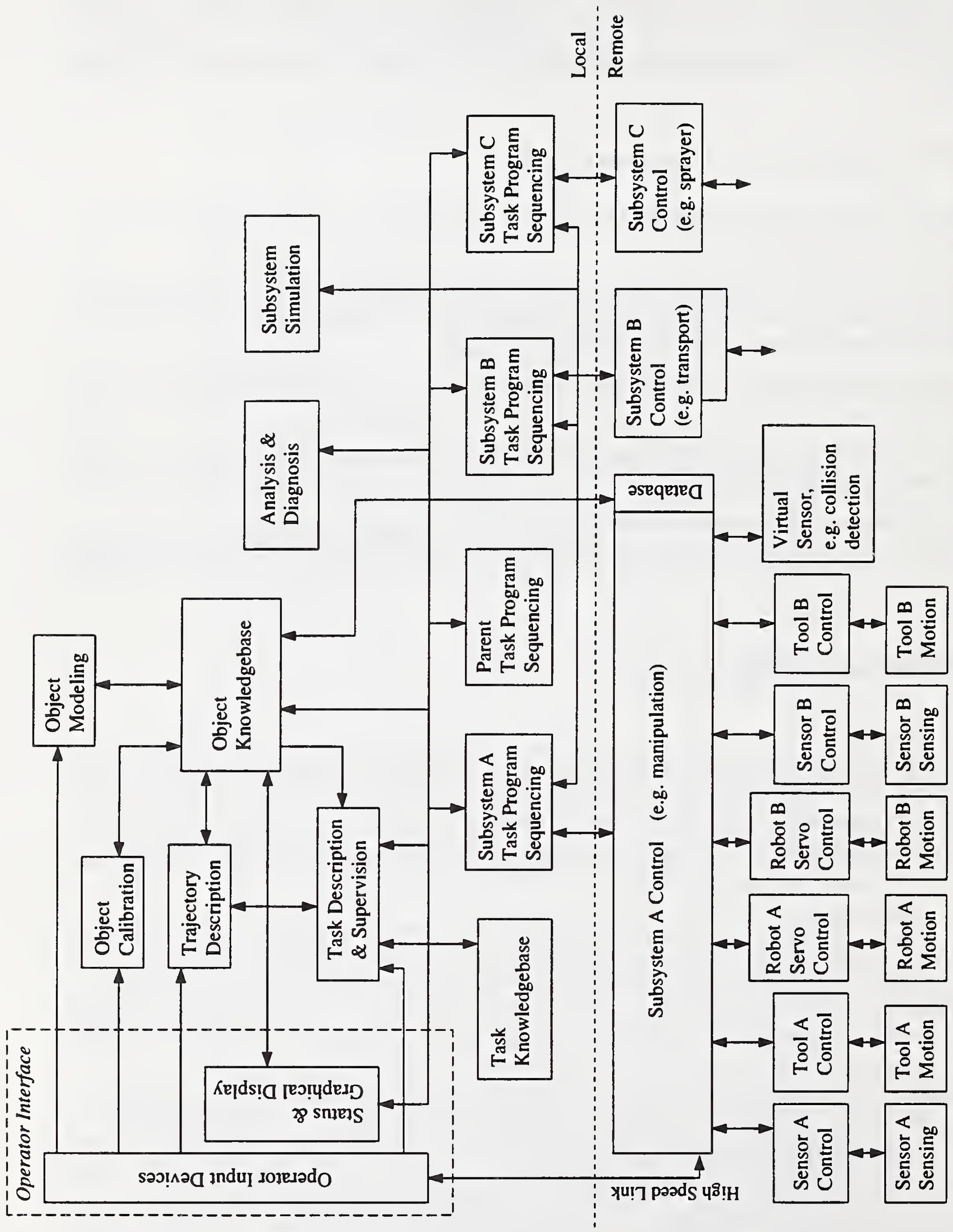

Figure 3 – Telerobot architecture for aircraft maintenance and remanufacturing

programs is achieved by direct communication between the subsystem task programs and/or through communication with a parent task program which communicates with the subsystem task programs to coordinate their control.

The generic architecture actually has separate hardware and software architectures since for different implementations, software of a specific functionality may reside on different computational hardware. For example, servo control software could reside on a special servo control board or on the same cpu board as task level control. The software module has a clear functionality, but where it is located is application dependent.

The NGC terminology for components is expanded here to separate components into three types of components: architecture components (AC), hardware components (HC), and software components (SC). Architecture components are the components consistent with NGC which are not hardware or software specific, but are functionality specific, and describe the application architecture. Hardware and software components are used in this report to specify the unique hardware and software modules of the system. This distinction is made because it is desired that the components be replaceable and software can be replaced independently of hardware and vice-versa. The architecture components are described in Annex A.

6.1 Hardware Architecture

The hardware architecture is shown in figure 4. The hardware components are separated

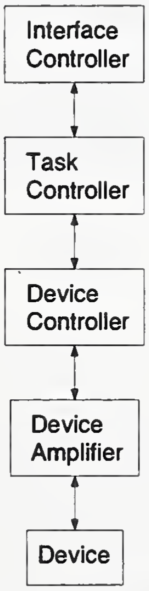

Figure 4 – Hardware Architecture

into physical hardware items that might be purchased. Hardware compatibility is a critical feature of an open architecture controller. Hardware compatibility implies physical connection between pieces of hardware. The connection can be communication lines such as serial, parallel, and ethernet cables, and the backplane which cards can plug into. These connection standards are not separate components, but are features between hardware components to make them compatible in an open system.

A common backplane for computer cards to plug into is a critical feature for hardware compatibility. Candidate backplanes for standardization include VME, ISA, EISA, and VISA. The backplane must support multiple processors and have sufficient throughput. It is important that the backplane specification is rigorous, complete, and unambiguous. A well specified backplane allows suppliers to develop boards for the backplane knowing the constraints for a board and for each pin, thereby ensuring safety for the hardware. The allowable power per board needs to be specified. Each pin that might be used by multiple boards needs to be well specified. If a pin is not well specified, then a supplier may not know how another board in a system might use the pin and therefore not be able to guarantee safety if the pin is used.

To allow a spectrum of common architecture controllers, multiple backplane options should be allowed, corresponding to lesser capability inexpensive systems and greater capability more expensive systems. Perhaps one low end backplane and one high end backplane should be selected. As with the NGC concept, the backplane is not specified here. The controller developer selects the backplane from a standard list for their specific profile. It is felt that a small number of backplanes will emerge. It is felt that presently the VME backplane is the desired backplane for high end, more expensive, systems and ISA and EISA for lower end, less expensive, systems.

The VME bus meets the backplane criterion the best of the backplanes considered. It specifies the allowable power draw per board. It's P1 bus is completely defined. The P2 bus has some defined lines and some lines left open to the user. The lines left open to the user must be further specified for use in an open architecture system. The other backplanes are less expensive and could be used in less demanding, or diverse, applications than the VME bus.

The hardware components are separated into types of hardware components as described in Annex A. There may be one (e.g., interface controller) or more (e.g., device controller) hardware components in the system for each type of hardware component. Also, the hardware component types apply for various device types including tools, sensors and manipulators.

6.2 Software Architecture

The software architecture is separated into functional types of software modules, the software modules themselves, and the application programs. Software modules are described in this report as modules, components and agents. The software modules, or components, can actually be aggregates of multiple software modules which collectively have specified responsibility, input and output. Modules and components both imply a software entity with a specific responsibility and inputs and outputs. The term agent also implies a software entity with a specific responsibility and inputs and outputs, but it also implies that this software entity runs as a separate thread of execution. An agent is likely to be an aggregation of software modules running as a separate thread of execution with a specified interface. Agent based systems have the benefit of being highly modular and reconfigurable with easily replaceable individual agents. Consistent with an NGC architecture, the software modules can be implemented as agents, i.e., as separate threads of execution. But, agent based systems have not been demonstrated sufficiently in real-time applications to justify a requirement for the use of agents. Therefore, the software modules of the system will be described as components with well defined responsibilities, inputs and outputs, but with the implementation details, e.g., the use of agents or not, left to the application system implementation.

The software module types and organization are first described followed by their interaction for supervisory and shared control and then by descriptions of the software modules as components.

6.2.1 Software Module Functional Types

The software architecture is shown in figure 5. Figure 5 distinguishes the types of software

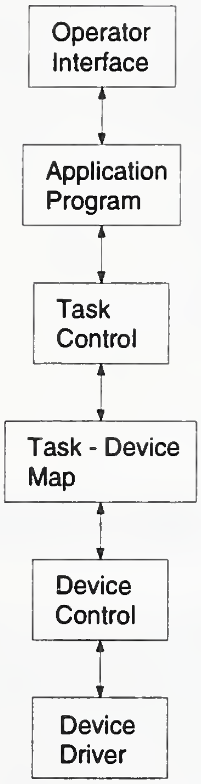

Figure 5 – Software Architecture

modules, or equivalently, components, for execution of an application program. A second figure, 6, shows how software modules are grouped and communication is constrained. The ovals indicate which components can communicate directly with each other; modules within the same oval can communicate directly. It is a goal to separate the different types of software components and specify their interfaces so that these can be developed independently. Libraries of application programs, macro commands, or task control modules could then be selected, perhaps purchased, as needed for a specific application.

The components of the architecture correspond to the components shown in figure 1, but the modules of figure 5 indicate the modules for task execution whereas the modules of figure 1 show components for both development and execution. The operator interface together with the application program represent the Application module of figure 1. The task control represents the Interpreter, Trajectory Planner and Controller and the task-device map represents the Inverse Kinematics of figure 1. The device control together with the device driver represents the Servo

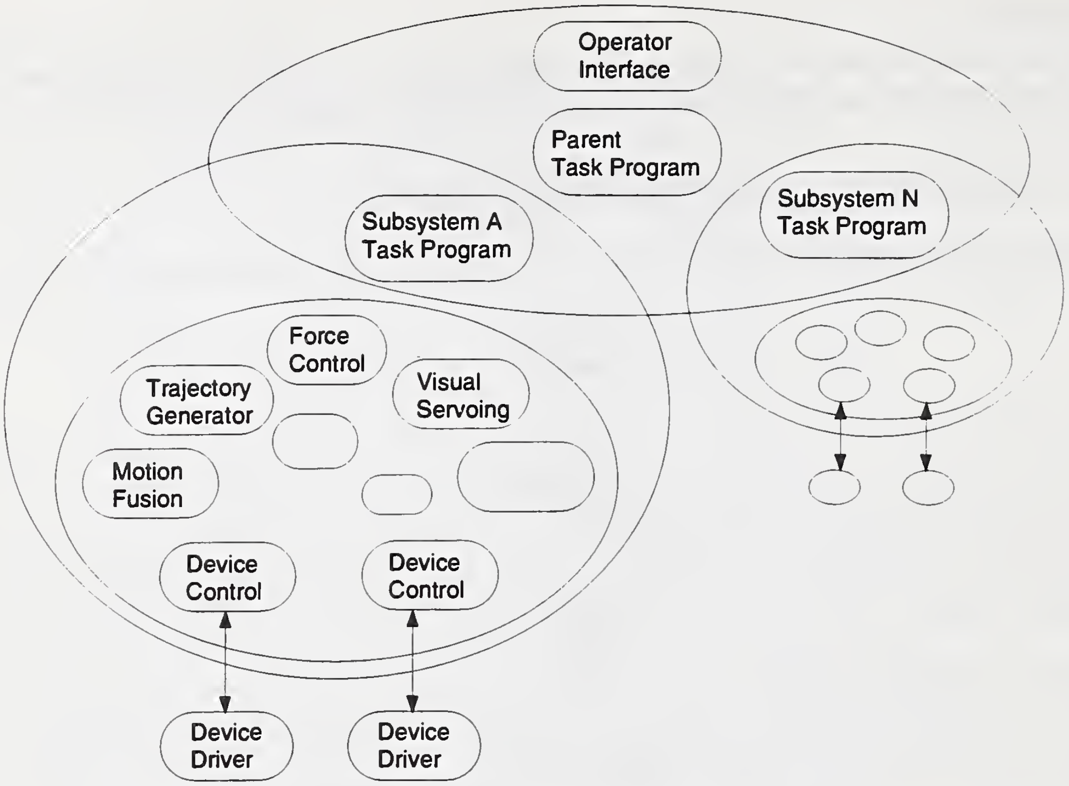

Figure 6 – Software Grouping

Control of figure 1. The language is an integral part of the architecture but is not a component since it does not process information. The architecture does not specify a translator module although one would be needed if the language for task description is different from the language of the commands sequenced in task execution. The macro commands and task program editor of the architecture represent the Tools component of figure 1.

The groupings of software modules given in figure 6 are functional. Where the specific software modules reside will depend on the profile of the system selected. The options for mapping software module types onto the hardware is given in figure 7.

The types of software modules that will be in a controller are described below, followed by descriptions of the actual software modules.

6.2.1.1 Operating System

The operating system is not one of the modules of the architecture. Rather it is a common service to the modules. There is likely to be separate operating systems for the planning and real-time control parts of the system - running on the interface controller versus running on the task controller computers. Standard operating systems would be very useful for development of an open architecture controller. Then all software modules could be developed and independently tested against given versions of an operating system. This would simplify software integration. If specific operating systems are not specified, then the constraints on the allowable operating

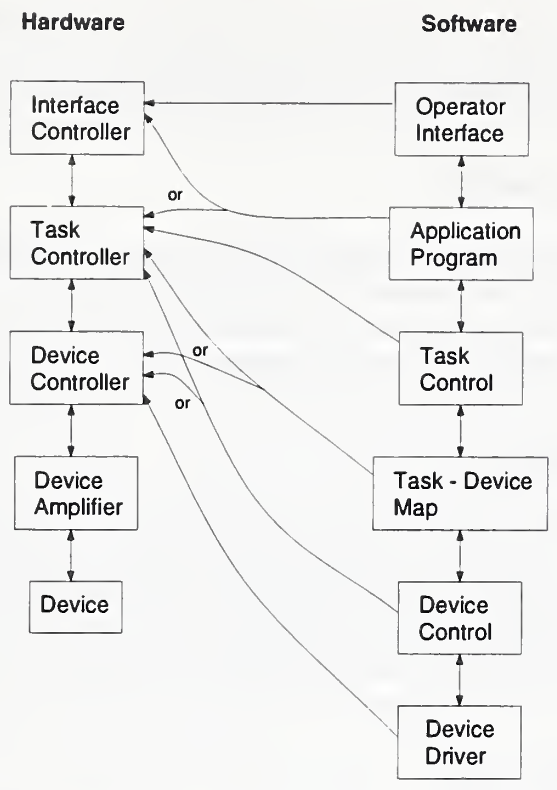

Figure 7 – Software to Hardware Map Options

system options should be specified, e.g., POSIX compatibility. The selected operating systems will be part of the architecture profile.

6.2.1.2 Operator Interface

The operator interface is the group of software which controls the inputs and outputs to the operator. This includes interaction with the application developer and the operator. The operator interface may be implemented in various forms, but a goal is to have a common method of interacting with systems across multiple applications. There will likely be multiple common interface methods, e.g., iconic systems with graphics simulation, or simpler (and less expensive) ASCII based inputs.

The operator interface software will run on the operator interface computer hardware. In some cases the operator interface computer will be the same system as the task controller hardware. This computer would then have to support the operator interface and task control software systems.

6.2.2 Application Program

An application program is the stored program which, when executed, will perform a task. An application program may consist of subsystem task programs and a parent task program. A subsystem task program can have multiple threads of execution, e.g., one for task control and another for status update to the operator interface. Application programs consist of sequences of macro commands and a limited set of conditionals and math operations.

6.2.2.1 Macro Commands

Macro commands encapsulate an algorithm which provides a type of capability, e.g., free motion command or grinding command. Macro commands will have a given set of parameters, but may have various internal implementations. Macro commands are automatically decomposed and translated into commands to control the task level control.

6.2.2.2 Task Control

Task control occurs when the task programs are executed. The task programs make calls to the task control modules. There will be many task control modules including force control, trajectory generator, visual servoing, monitoring, and motion command modules. By specifying the interfaces of the modules, modules can be acquired from different sources. The various sources of motion will generate motion commands which the motion fusion module will merge into a command to the device to be controlled.

6.2.2.3 Task-Device Map

The task level command is transformed into the actuator coordinates, e.g., joint angles, velocities or torques, with a task-device map module. A separate module is used to transform measured data from the device coordinates to the task space coordinates. This module is mechanism dependent. Sensor commands also go through a task-device functional module to transform the command to the coordinates of the sensor, and when read, from the sensor frame to the task control frame.

6.2.2.4 Device Control

The device control software modules provide the control of the axes of the mechanism or interface to sensors. It also provides the interpolation of setpoints for the device servo control since this is likely running faster than the task level control and thus has multiple cycles between commands from the task level control. Servo control of joints would be done by device control software modules. The device control software modules might reside on the task controller or device controller hardware. For example, a controller may allow joint servo control software to reside on the task controller board with the task control software. The device controller hardware might then just be a D/A card. Alternatively, a servo control card could be used for the device controller hardware which would have the servo control device control software on it. The device control module communicates with the power interface to the device, e.g., PWM commands to PWM drives, analog commands to analog drives, signal interfaces to sensors.

6.2.2.5 Device Driver

The device driver software is hardware dependent, residing on the device controller hardware. This component sends commands and receives status from the device amplifier hardware, e.g., voltage or PWM signals.

6.2.2.6 SW Architecture for Supervisory and Shared Control

A unified supervisory and shared control telerobotic system has the same architecture for all modes of control: teleoperation, shared control and supervised autonomy. The fundamental system provides task description and task program sequencing. The commands in a sequence can imply autonomous execution or a mix of autonomy and teleoperation inputs. There are two basic paths for operator inputs. The inputs can be incorporated into parameters in the command path from the operator interface to the task execution system, or hand controller inputs from the operator can be treated as sensory input to the real-time task level control. In both cases they have similar form as other types of information. The system is therefore essentially an autonomous control system which allows operator inputs during execution.

Different components of the system might run synchronously, asynchronously or upon request. For example, in the real time control, the closed loop control components might run asynchronously at different rates, reading available data, and producing data to be read. Slowly changing information can be computed at a slower rate than it is used. Alternatively, these components could be synchronized and called in a given order. The task planning components will not be called at the high rates that the task control components are run. Therefore, they could more readily be implemented as agents, responding only when their services are needed.

There may be multiple sources for motion of the tool, and therefore the manipulator, including hand controller, trajectory generator, and closed loop sensor based control such as force control and proximity control. Motion commands from each of these sources can be generated by specific software components associated with the motion source. These motion commands then have to be merged. This merging is done by the motion fusion component. There are many ways that motion can be merged, or fused, with motion commands of various types, e.g., disturbance forces, incremental motion, velocities and absolute positions. The motion source components therefore have to generate motion commands which are consistent with the motion fusion component input types.

7 Interface Environment

The UTAP architecture is a modularized arrangement of control services. As a result, a modularized system reduces complexity and makes it easier to understand, design, and implement the system. The complementary result of the modularization of a system into components are *interfaces*. An interface provides access to a module's services where each interface is defined by a language that specifies the tokens (or keywords), syntax (or format), and semantics (or legitimate values and interpretation) that are acceptable to a module. Indeed, one could have several interface languages to the same module of computational services. The goal of a generic interface is to unify similar computational services under one, general-purpose, access mechanism supporting a wide range of uses.

The major observation within the UTAP Interface Environment is that an interface is composed of two elements, a language and a protocol. This observation can be represented by the following equation:

$$Interface = Language + Protocol \tag{1}$$

Which is equivalent to saying that an interface is defined as "What-to-pass" plus "How-to-Pass."

In order to realize the hard real-time processing demands of motion control, one requires that UTAP interface languages must be efficient and allow timely transmission and interpretation of data. A modeling schism develops attempting to meet the desire for generality and the requirement for performance. A more expressive language is desirable but suffers the penalty of an increased performance requirement. The UTAP Interface Environment is framed by the assumptions made in order to resolve conflicting notions of interface definition. (See Annex G for further discussion of these issues.)

NOTE 1 – Ultimately, the following assumptions were made for this Working Draft Standard Document. The first assumption was to focus on what-to-pass, not how-to-pass. The second assumption was to minimize complexity and adopt a simple definition style. A simple language strategy would appeal to a greater audience. Initial attempts at an elegant software solution were confusing and drew attention away from the focus of the problem - defining the language primitives. The third assumption was that a "published" interface would be necessary. A published interface would require minimal interpretation and allow shared memory schemes. The fourth assumption was to allow both measurement units and computer representation to be adjustable. The environment would explicitly define message primitives for different units and representation. The fifth assumption was to provide for symbolic manipulation of data, in that, although the message definitions were in a raw format, textual information would be required also.

The definition of interfaces consists of two elements: *Configuration* and *Language Framework.*

Configuration deals with naming, system identification, narrowing the scope of the problem through labelling, and system scaling. Naming includes acronyms, message naming conventions and communication channel naming conventions. Configuration includes classification, resolving duplicate module types, dynamic configuration and attachment of a protocol to channel. A service directory is associated with each module that describes the permissible set of messages into/outof the module. This capability allows scaling of the system.

The Language Framework covers *Information Models* and the *Interface Language.*

Information Models define the data representation within the messages. A substantive information model is required for interoperability. One could define everything as tokens, but this offers little in helping with the standardization process. The Information Model includes 1) domain-independent items or generic data definitions; 2) feature-based definitions such a geometry, topology, shape, and patterns; and 3) object knowledge. Object knowledge covers the devices, parts, modules, and general system state information. Object knowledge is defined with attributes, and access to information is through query/response connection.

The UTAP Interface Language was defined as a set of messages. The C/C++ language was used to define messages. There is a trade-off between interface language complexity and performance. The distributed and real-time nature of the UTAP predicated an explicit, simple approach to defining messages. The UTAP message defining style uses #defines to enumerate message name and id, plus gives data structures to each message id. The information models (data declarations) and messages were defined within C/C++ header files. The information models and messages are compilable. A more abstract Application Programming Interface is defined and was derived by running a filter on the message definitions.

Generic messages were defined that are applicable to all modules in the UTAP architecture. Mode and state change commands are covered by the generic messages. Such state change commands include: start, halt, hold, resume, suspend, etc. Extensibility and customization are provided with the MACRO, and PLAN set of messages. Synchronization of messages is provided with the BLOCK and EVENT set of messages.

The UTAP framework provides for these major styles of messages – sensor/effector control and query/response. The UTAP sensor/effector (S/E) control interfaces apply programming concepts from servo control, programmable input/output and the programming format RS274 [EIA274]. The S/E control interfaces divide communication into 1) mode and 2) action messages. The mode messages provide for event sequencing (e.g., start, halt, abort, etc.), set-up, algorithm selection (e.g. PID, FEEDFORWARD, etc.) and provide for loading control parameters. The action messages either write a command or initiate a sensor reading. Action messages treat communication as clocked data flow. Query/Response (Q/R) interfaces adopt a similar strategy but one generally assumes one cycle per clocked data flow. The Q/R data can be of the form of a query message from the superior to the subordinate, or as a reading from the subordinate to the superior and/or Object Knowledge module.

7.1 Viewpoints

Some interfaces do not need to have an innate understanding of the control domain and will be merely performing symbolic manipulation of the interface data. For example, an Object Knowledge Base or Operator GUI do not need to understand the application in order to store/retrieve or display the information. Instead, these modules must have a systematic (and symbolic) means of receiving system information and capabilities, and then organizing this information for either the user or other modules in the system. For example, I as a user may wish to override the feed rate for a particular task if I observe chatter. The GUI cannot understand why I'm changing the feed rate. Instead, the GUI may have limits on the acceptable range of numeric values, and pass the new value to the control system which makes the determination for the validity of the new data value.

Likewise, when one module requests the value of the feed rate from the Object Knowledge Module, the data manager doesn't need to know the purpose of the feed rate, but rather, it needs to know its computer representation (double), its range of legal values, and possibly all the users of this information throughout the system.

The UTAP interfaces will provide a capability, tasking, and data framework for the modules that may only require symbolic manipulation of data. The capability framework specifies what resources are in the system, e.g., robots, tools and sensors. The tasking framework will provide the necessary knowledge about how the capabilities can be used. The data framework provides an all-encompassing description of the potential data that the system has at its disposal. It is foolish for the system to pass every conceivable variable to the Object Knowledge Base. Instead, we will assume that configuration of data posting, data viewing and data modifying is possible.

For instance, various configurations could define what values are periodically posted to the object knowledge base, what values are visually presented to the user, what presentation style the user prefers to view data, etc.

8 UTAP Information Models

The UTAP applications operate on such parts as wings, fuselages, and other plane related parts. These parts can be described as a combination of geometry, topology and shape to derive UTAP features. These features are used to identify the focus of attention for the tooling operation. For the initial phase of the UTAP, features will be described as simple shapes that are filled by motion patterns.

The UTAP framework will use information models to describe part features and system attributes. Currently, the information models include generic types, part information models, and system data definitions. Generic information models cover domain-independent types. The generics include basic data types including: mode directive, generic_directives, user type, mode states, results, and a state_type. The part information models define measurement units, representation units, features and object attributes required of the system. System data definitions are intended to cover sensing and control attributes.

The sum of these information models describe the Object Knowledge and are preliminary. The files generic_defs.h, utap_info_model.h and utap_data_defs.h in Annex H.3 present the current state of these definitions. Presently, the feature-based information model is relatively modest. We have provided for an evolutionary path to allow for growth of potential part shape geometries. The ISO STEP Part for Geometrical Shape and Material Information Models [STEP42] covers a more complete range of data modelling.

NOTE 1 – A translation from EXPRESS Part Model into a C++ language information model can be done, and was done to derive the current set of data definitions. To provide for a broader set of part description, the STEP Part 42 geometrical models could be substituted for the current data definitions – but is beyond the scope of the current level of effort.

8.1 Shape Geometries

For the sake of clarity only the range of part shapes that are foreseen within the scenarios will be addressed. These parts are of course a small subset of the realm of potential parts shapes. The major assumption to the current definition of the UTAP part geometry is that the operator will define or choose the workpart geometry from a set of prescribed shape models.

The UTAP interfaces have a preliminary geometric shape model that describes the shapes required within the application scenarios. Such shapes can be one, two or three dimensional. The shape dimensionality specifies the geometric form of a topological or geometric entity. Edges (curves), Faces (surfaces) and shells(volumes) have dimensionality of 1, 2, and 3 respectively. By convention a Vertex (point) has dimensionality of 0. These geometric shapes required of the UTAP interfaces include:

– 1D - surface, planar or curvilinear edge

- 2D - rectangle, circle, polygon or connected edge list
- 3D - box, cylinder.

The features can be embedded within each other. This capability allows us to define circular/rectangular obstacles within our workarea feature.

8.2 Patterns

A type of motion within the work volume will be termed the motion pattern. Some patterns are merely shorthand notation for a larger set of motions. For example, a raster motion sweep can be composed of a set of linear motions. But, it is more intuitive to the operator (and programmer) to define a raster pattern within a rectangle workarea. Patterns can be shape fill patterns or edge patterns. One is either applying a motion pattern to the face of a part or to the edge of a part.

The edge patterns are:

- exact or within some tolerance along edge
- sine or square-wave weave (e.g., for arc-welding) The fill patterns are:
- horizontal and vertical raster
- orbital type motion
- dithered or chaotic motion
- concentric circle fill.

Of course, these definitions are not complete but appear to handle the task scenarios. New pattern definitions can easily be added by a systems programmer as the need arises.

8.3 Features

A UTAP feature is a combination of a geometrical shape and pattern to describe the motion applied to that shape. For the UTAP application domain, the primary features will be pattern motions within faces of different geometrical shape - e.g., flat surfaces with rectangular and circular features or, curved surfaces with conic features. In this case the faces (and their constituent edges, vertices, surfaces. etc.) are the primitives that can also be operated on. Given the shape we must then describe the motion pattern that will be applied to the feature. Thus, we define a feature with the following equation:

$$FEATURE = shape + pattern \tag{2}$$

Features are constructed using the following base definitions:

- GEOMETRY: gpoint, vector, pose, transform, arc

- TOPOLOGY: tpoint, vertex, edge, edge_list, loop, face
- SHAPE: box, rectangle, helix
- PATTERN: edge_pattern, blend_pattern, face_pattern, shell_pattern.

The part shape geometry determines the work volume. The topology is used to define boundaries. Shape is derived from geometry and/or topology. Most of the application scenarios involve tooling the surface area or face of a part. To cover the surface a series of motion patterns will be required.

9 Integration and Configuration Management

The UTAP architecture emphasizes telerobotic control. Because of this, the UTAP architecture is divided into a REMOTE teleoperated partition and a LOCAL motion and tooling control partition. Although the REMOTE topology of the UTAP architecture is a static arrangement (i.e., they is only one instance of many of the modules), the LOCAL topology will vary between actual systems. For the LOCAL partition, the UTAP architecture describes a topology framework for composing modules. Identical modules can exist as subordinates to the same superior. For these modules to be configured in a complete topology, an identification or naming convention is required. With a naming convention, a directive will be sent to the proper subordinate.

A classification framework helps bound the range of module capabilities and to provide for a smooth evolutionary path. For comparison, the term "printer" - although descriptive - can be vague. One can have a color printer, a dot matrix printer, a laser printer, ad infinitum. Without a classification framework, one cannot accurately determine the expected capabilities of the modules. Before one can define interfaces one must categorize the range of modules in a UTAP system. In turn, proper categorization of UTAP systems will provide a more coherent framework for defining the interfaces.

9.1 Identification

The UTAP describes an architecture that can vary in size and complexity. For some of these modules, only a single-instance of the module exists in a system. These modules include OI, OM, OC, TD, TDS, TK, PTPS, SGD, AD, and SS. For the remaining modules (i.e., TPS, TLC, SC, RSC, TC, and VS), multiple-instances can exist in the system simultaneously. The variability of the number and scope of the module members means the system architecture can vary. To quantify the size and scope of the architecture, one is required to 1) enumerate the active single-instance modules in the system and 2) identify and categorize the multi-instance modules.

To define the multi-instance modules, the identity, the grouping, and the relationship of modules must be defined. The multi-instance modules must be declared and linked to the relevant superior/owner/parent. Multi-instance modules require a unique naming convention. The proposed syntax for module naming is the following:

```
        module_list:=  module { module_list }
        module:=  module_name:identifier
```

This syntax enables a system to be described as a tree. Some modules are capable of controlling multiple subordinates of the same module type. Each subordinate of identical module type must have an instance identifier. For example, there can be multiple Task Level Control modules i.e, TLC:A, TLC:B. In turn, each of these modules is capable of controlling a subsystem, e.g., TLC:A:TOOL:A, TLC:A:TOOL:B.

9.2 Classification

The UTAP architecture contains a list of 20 modules. This architecture has the potential to describe a broad range of systems. The realm of possibilities should be narrowed to allow ranges of compatibility. To achieve compatibility, one needs to attach labels to identify the types of modules. Table 1 describes a naming scheme that classifies modules with type labels and illustrates the operational relationship among modules. Those types that have a preceding asterisk will not be considered in the UTAP at this time.

9.3 Configuration File Format

The System Configuration Files will be responsible for defining the architectural tree. The System Configuration Files are a combination of 1) the multiplicity of modules, and 2) the classification labeling scheme. Examples are given in Annex annex:example. The purpose of the tables is to assist in determining module interconnections and interface naming convention. The exact format of these file is not currently defined. There is great potential for the configuration file that will not be addressed here.

9.4 Module Specification

The conceptual model forms a framework for the required functionality required of a UTAP module interfaces. The UTAP conceptual interface framework will be described as with a set of component units. Figure 8 illustrates a conceptual model of the UTAP module and the component units. In this conceptual model, the UTAP will adopt the strategy that a module must publish a SERVICE PROFILE of accepted messages and postable data. The SERVICE PROFILE is the umbrella under which a module declares its capabilities. The SERVICE PROFILE unit contains a slot for defining the timing of the module. A conceptual module contains a CLI or command line interface unit that receives either transmitted command messages or has the ability to read programs or commands from disk. The CLI is responsible overseeing the set up of modes and PARAMETERS for a given module. The POST unit within the conceptual model is responsible for maintaining the module output updates. These output updates are periodically sent to either the Object Knowledge module or the SUPERIOR module. The PROG MACROS unit allows aggregating and naming of parameter or command sets.

Annex C contains boilerplate Service Profile checklists for the remote modules.

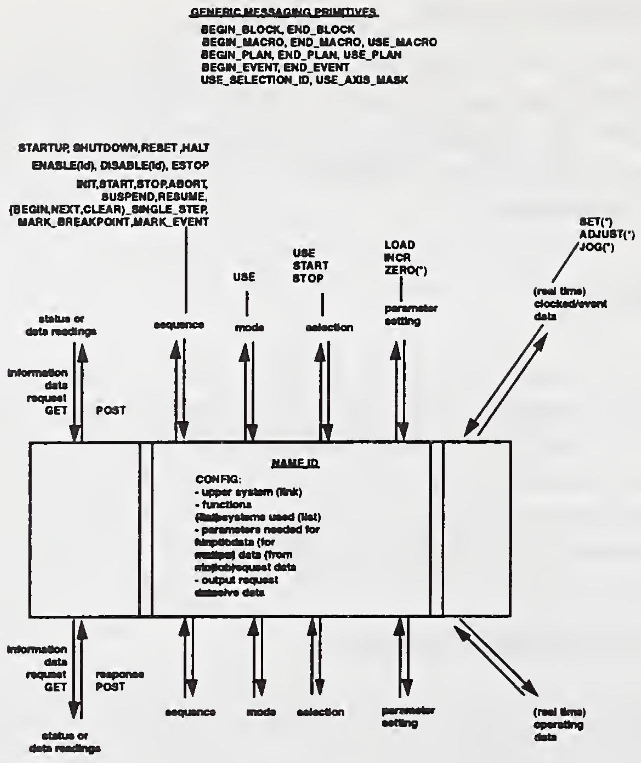
GENERIC MESSAGING PRIMITIVES
BEGIN_BLOCK, END_BLOCK
BEGIN_MACRO, END_MACRO, USE_MACRO
BEGIN_PLAN, END_PLAN, USE_PLAN
BEGIN_EVENT, END_EVENT
USE_SELECTION_ID, USE_AXIS_MASK
STARTUP, SHUTDOWN, RESET, HALT
ENABLE(Id), DISABLE(Id), ESTOP
INIT, START, STOP, ABORT, SUSPEND, RESUME, (BEGIN, NEXT, CLEAR)_SINGLE_STEP, MARK_BREAKPOINT, MARK_EVENT
USE
USE START STOP
LOAD INCR ZERO(*)
SET(*) ADJUST(*) JOG(*)
status or data readings
information data request GET
POST
sequence
mode
selection
parameter setting
(real time) clocked/event data
NAME ID
CONFIG:
- upper system (link)
- functions
subsystems used (list)
- parameters needed for input data (for
data (from
request data
- output request
data
information data request GET
response POST
status or data readings
sequence
mode
selection
parameter setting
(real time) operating data

Figure 8 – Module Specification Model

9.4.1 Scaling

The system should allow scaling. The set of UTAP messages is quite extensive. It is not expected that all modules should accommodate every interface message. Further, some systems will specialize in certain aspects of control or sensing, and completely ignore some aspects of a UTAP module interface. The goal of the UTAP was to scale options through the message list. Many of the messages could be combined under a broader message category but this creates a problem. How do you say that I accept this message but not a certain part of the message? It was felt that scaling would be best accomplished if maintained under a single concept of reference.

The UTAP interface definitions are designed to remain constant whatever the system capability. The UTAP module SERVICE PROFILE is defined to provide a scaling mechanism. For each module, the SERVICE PROFILE describes the set of acceptable UTAP messages and data posting capabilities.

9.4.2 Timing

The timing deadline element within the interfaces will be done in a worst case manner. The module will specify the worst-case time duration that it can receive expects new message. The modules must publish this value.

10 UTAP Interface Framework

10.1 Interface Types

Not every interface in the system is identical conceptually. Communication interfaces will be categorized into the following groups: 1) control for superior/subordinate command-status interfaces, 2) query/response, and 3) peer-peer event synchronization.

10.1.1 Control Interface Type

A superior/subordinate control interface type applies to either Sensor or Effector (S/E) behavior. Each control interface is part of a larger chain of command. The objective of the control interface is to make the subordinate do something for the superior. The subordinate may be a simple slave that simply obeys the orders from the superior and translates these instructions into some machine physical format. The subordinate may contain some intelligence and add some functional transformation of which it is the expert. See Albus [RCS] for more insight into this command and control theory. The UTAP control interfaces will adopt a format that draws from concepts used in Servo Control, Programmable I/O chips (PIO), and the RS274D language.

The UTAP control interface mimics servo control with communication from a superior to a subordinate module treated as clocked data flow. Of note, the clocked data flow may only last one cycle. The clocked data flow can be either control commands or status readings. For control commands, response to the command is not an answer, but a servoing action and status report. For status readings, response is either a status report or a sensor interpretation.

The UTAP control interface strategy adopts command, status and mode concepts of PIO chips. A programmable I/O chip (PIO) has operational modes and parameters that must be initialized before the chip is functional. Further PIO chips allow for combinations of selection modes. Selection vectors are of extreme relevance to teleoperated robotics - for example, the application of force control in one axis, while using position control in the other axes. UTAP interface format applies the PIO programming paradigm requiring to first initialize the subordinate with the appropriate mode and control parameters, and then initiate data exchange. The ability to combine modes and load parameters creates the potential for errors - either over or under specifying of the desired control/sensing strategy. These error cases have associated UTAP messages.

The blocking sequence and synchronization concept of RS274 were used and extended to accommodate other needs. Although the UTAP set of messages for BLOCK, MACRO, PLAN, and EVENT are primitive computer language constructs, they are helpful in reducing the complexity of an interface. Each of these language primitives is discussed further in a later section. The BLOCK messages allow for synchronizing concurrent events. The MACRO messages allow a series of mode and parameter settings to be grouped and named. This allows for easy context switches among operational modes. The EVENT messages are designed to augment the BLOCK messages and offer more robust synchronization of operation. The PLAN messages are for grouping and naming a set of data commands, e.g., naming a SAFE motion or zone.

10.1.2 Query-Response

The query response interface is more closely associated with state knowledge or sensor reading updates. One form is the client/server which provides a dialogue or question/answer interface. The client asks the server (in many cases a subordinate or expert) to periodically post status or state information. This posting can go to the superior or to the object knowledge module.

Obviously, one doesn't want every conceivable piece of system state information flowing through the system at every clock cycle. One would prefer that under certain circumstances, relevant state information is posted in the timely manner desired. For example, under normal operation, it would be desirable to post the current position as status every 10 milliseconds. For gain tuning, one may require position readouts every millisecond. Under maintenance operation, it might be desirable to post the current position and encoder readings so that a problem can be tracked down. The interface must be flexible and allow a range of state information to be posted.

The ROUTE data structure defined below was intended to provide a contextual-based mechanism for posting state information. A module would receive a get-info query and then post the desired state information. Depending on the type of get, the state information could be posted once or periodically updated. The same mechanism can be used to read state information data from the Object Knowledge module.

```
struct ROUTE {
        enum {  _STATUS        = 1,              // post response to questioner
                _WRITE_TO_OK   = 2,              // posting response values to ok
                _READ_FROM_OK  = 4,              // read from ok
                _DELTA_OFFSET  = 8,              // use data as delta offset
                _ALTER         = 8,              // alter cmd dx,dy,dz,rx,ry,rz
            } type;                              // Bitmask for response dest
        int times;                               // -1= continuous, 0=stop, 1=1,...
        TIME update_period;                      // frequency of update

    };
```

The range of potential Object Knowledge attributes is formidable. As a basis, the following generic attributes have been designated. These attributes cover both sensor/effector control and part modeling information. The baseline UTAP data dictionary of parameters is given in Figure 9.

acceleration	jerk	pressure
attribute_name	luminance	roughness
device_units(e.g.,encoder_ticks)	mass	temperature
flow	material	time
force	material_name	torque
geometry	object_name	velocity
hardness	orientation	viscosity
humidity	pose	others
Jacobian	position	

Figure 9 – Object Knowledge Parameter List

The concept of max, min, avg, real (current), desired, last, and timed historical reading (e.g., 2nd to last) will be used as *attribute modifiers* within the message interface. Thus, one can get and post desired position and real position. The attribute in this case is position, and the modifiers are desired and real.

10.1.3 Peer-to-Peer

Peer-to-peer may be necessary for synchronization of modules. Cases such as awaiting the completion of fixturing by an operator or awaiting the completion of a tool change before moving are examples of synchronization events. Synchronization of this type can be avoided by synchronizing events at a higher level in the architecture. We will assume this can be done, and will not address peer-to-peer synchronization within the UTAP at this time.

10.2 Syntactics

Interfaces will have the following naming conventions. The C preprocessor directive #define will be used to define message names and assign a unique system numeric id. Each message name will be in capital letters. Each message name will be prepended by a US for Unified System. The US part will be followed by the module name - unless the message is a generic message - e.g., US_ModuleName. Then, the actual message name will follow - e.g., US_ModuleName_MessageName. The data type naming convention will use lower-case letters and in general merely append a _msg_t to signify message type, e.g., us_modulename_messagename_msg_t.

Table 2 gives a summary table of contents for the message numbering.

Annex H summarizes the set of messages. Within Annex H.3, the file utap_interfaces.h contains the message id and associated message structure. Below are two examples from this file. The presentation style has a #define message id preceding each message structure. So far, there are approximately 250 messages.

```
#define US_HALT 102
struct us_halt_msg_t {
  int msgid;
} ;

#define US_AXIS_SERVO_LOAD_PID_GAINS 210
struct us_axis_servo_load_pid_gains_msg_t {
    int msgid;
  double p;
    double i;
    double d;
};
```

10.2.1 Variable Length Arrays Resolution

One of the difficulties that arises defining interfaces concerns the problem of handling variable length arrays. Unless one rejects the notion of flexibility, an interface should not preordain a fixed array size for any interface. One would find passing 7 joint values to a 3-axis mill less than intuitive. Generally, array pointers are used to overcome this problem.

The UTAP interfaces *shall* use the following strategy: 1) if necessary, declare the degrees of freedom as a mode parameter, and 2) reference data array information indirectly into a heap mechanism (i.e., a zone of memory in which multi-linked nodes of variable size are allocated) that

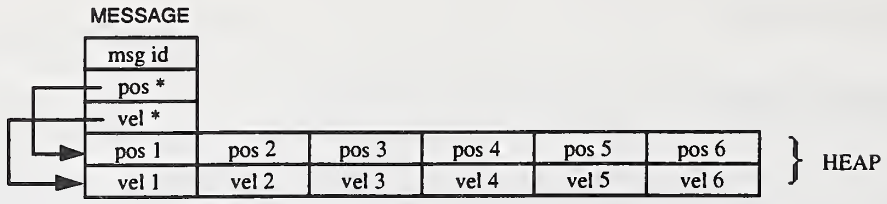

Figure 10 – Heap Applied to Message Handling

follows the message. Figure ?? illustrates the concept when passing an array of joint positions and velocities to a 6 DOF robot. Should one pass to a 3DOF machine tool, the message would still have the position and velocity contents, but the heap would only contain three elements for each field.

Overall, the message structure can be represented with the following equation:

$$MESSAGE = HEADER + CONTENT + HEAP \tag{3}$$

where the HEADER contains protocol or "how-to" specific information, the CONTENT defines "what-is" or the message information, and the HEAP contains the variable-length data contents.

10.3 Semantic Meaning

At this point, the exact semantic meaning of many of the UTAP messages has not been explicitly spelled out in English. For now, the intent and meaning of UTAP API messages *should* be apparent from the message name.

One simplification was the use a special keywords and a keyword convention to specify the semantic intent. The keyword convention provides consistent message naming which leads to easier comprehension. The UTAP naming convention follows a generic flow plan that categorizes control, data, parameter and mode message traffic. Figure 8 illustrates the flow of information and the naming convention relationships. The flow of traffic is divided into 1) control sequence, 2) modes, 3) algorithm selections, 4) parameter settings, 5) real-time data, 6) information requests and 7) information responses. The information flow is equivalent for superior as well as subordinate connections, except that there can be multiple instances of subordinate information flow. Although conceptually demarcated, the information flow would most likely require only one or two connections to the superior, and to each subordinate - one for command and possibly another for status. (The bi-directional arrows for many of the categories was used to convey the notion that one wire is for commands and the other wire is for errors or acknowledgment.)

The naming convention uses keywords to delineate mode/goal/state information. These keywords are embedded within the messages to categorize the semantic interpretation of a message. The keywords are grouped by type:

- MESSAGING (i.e., BLOCK, MACRO, PLAN, EVENT, SELECTION)
- SEQUENCING CONTROL: generics (i.e., STARTUP, SHUTDOWN, ENABLE, DISABLE, etc.)

- MODALITY: USE, START, STOP, COMPUTE
- PARAMETRIC: LOAD, INCREMENT, SELECT
- DATA COMMAND: SET, ADJUST, GET
- STATUS: POST

Annex C provides a module by module profile of the UTAP flow plans. These profiles are blank templates that can be used to specify the requirements of a desired system. These profiles provide a complete list of all the potential inputs and outputs of a module. The annex lists input and output entries by flow category. Some categories have cross-references to other flow plans.

10.3.1 Control Mode Sequencing

Most module control sequencing is done with generics. The sequencing generics are grouped by levels of operation - module operation, sensor/effector operation, and software operation. STARTUP, RESET and SHUTDOWN are module power-cycle sequencing operations. ENABLE DISABLE, HOLD, and ESTOP are sensor/effector power sequencing operations. INIT, START, STOP, PAUSE, HALT, ABORT, are software basic program sequencing commands. BEGIN_SINGLE_STEP, NEXT_SINGLE_STEP, CLEAR_SINGLE_STEP, MARK_BREAKPOINT and MARK_EVENT are generic keywords that deal with more advanced program execution. The only generic not commonly found is MARK_EVENT which is used as a reference marker for an EVENT primitive.

A typical sequencing operation consists of the following steps. First, the control sequence keyword STARTUP brings the module into a safe state. Second, one programs the module with the appropriate control, mode and parametric settings. Once programmed, the sensor/effectors are powered on with the control sequence keyword ENABLE. Finally, the software program is executed by issuing the control sequence keyword START. At this point, clocked data flow commences.

10.3.2 Keywords

For mode messages, the words USE, START, STOP, COMPUTE are used to convey the notion of parameter setting or algorithmic selection. The word LOAD and INCREMENT are used for parameter values. The words SET, GET, and ADJUST are used to denote a commanded action.

USE:
The USE keyword conveys the notion of modal or a mutually exclusive algorithm selection. Modal commands stay in effect until cancelled. The COMPUTE keyword is a synonym for the USE keyword and exists since some messages aren't as apparent with the USE keyword, e.g., USE_GREY_VALUE versus COMPUTE_GREY_VALUE.

START/STOP:
It was decided that the UTAP interfaces must support simultaneous multiple actions. The

terms START and STOP convey the notion for initiating/terminating simultaneous or multiple selections. Thus, one can START one or multiple algorithms when necessary. Then, the command STOP is used to discontinue the algorithm. For example, gravity compensation can be used to augment many servo algorithms. Thus START_GRAVITY_COMPENSATION remains in effect until terminated with a STOP_GRAVITY_COMPENSATION.

LOAD, INCR(EMENT), ZERO:
The LOAD keyword signifies parametric value setting. Load implies that one presets a parameter during initialization, or can dynamically change the value during clocked data updates. For example, velocity can be fixed or dynamically updated with each motion. The INCREMENT keyword is used to denote an incremental (or decremental) update to a parameter. For example, INCREMENT is useful for relative increases of desired velocity. INCR may be used as shorthand for INCREMENT. The ZERO keyword means to use the current reading as the origin. Thus, ZERO is often used to initialize a relative coordinate frame. The SELECT keyword is used with a parameter setting for choosing from a set of established parameters.

SET, ADJUST:
The SET and ADJUST keyword signify instructional command messages. These messages provide a goal for the module to achieve. The verb SET signifies a goal for the subordinate to attain or maintain.. The verb ADJUST is used to convey incremental or decremental changes. Such changes are useful for tuning the commanded goal state. Conceptually, this is equivalent to moving to a position, zeroing the position as a relative position and then moving a small relative offset - but ADJUST is conceptually simpler. A joystick +/- button interface is an example of interface that requires the ADJUST concept where positional change is relative.

GET:
The verb GET is used to initiate a response - either a status report, or a sensor reading. GET has a corresponding response POST. The GET is part of the Query-Response interface connection. Each GET message contains routing, attribute and modifier information. For example, the generic message US_GET_VALUE requires routing, an attribute and a modifier.

```
#define US_GET_VALUE 55
struct us_get_value_msg_t {
        int msgid;
        ROUTE r;
        Attribute_t items;
        Modifier_t modifiers;
};
```

The routing information (i.e., ROUTE) describes the return destination of the response. The Attribute_t entry contains the query identifier. For common attributes, the attribute information is explicit to the message name, e.g. US_ATTRIBUTE_GET_POSITION. The Modifier_t describe state information about the attribute, e.g. real, desired, max, min, etc. GET_LIST is available to query information about data collections.

POST:
The verb POST is used to convey the notion of an output reading. This output reading could be either a status report, or a sensor reading. Further, the output reading could be posted to the superior or to the Object Knowledge module. Within the GET, a field is set aside to designate the destination of the subsequent POST - either to the superior, Object Knowledge module or both. The POST messages use this information for return routing.

INPUT/OUTPUT:
Interfaces that support query or posting services and provide the same interface to numerous modules will adopt the following naming convention: for input and output, append either _INPUT or _OUTPUT to the server module name, e.g., OK_INPUT.

10.3.3 Designating Subordinate Selections

The UTAP architecture allows multiple subordinate modules to be controlled by one superior module. In Figure 3 for example, the Task Level Control module is controlling 6 submodules - pairs of robot, tool and sensor modules. Because of the existence of alternatives, a robot command issued to the Task Level Control can be ambiguous as to which robot the command is intended. Additionally, generic messages such as LOAD_SAMPLING_RATE are applicable across many of the subordinate modules. Thus, an interface mechanism is required to resolve this confusion. Naming resolution of identical module types was addressed within the configuration section. This section describes a message set to enable selections.

The generic messages GET, POST, USE_SELECTION and USE_AXIS_MASK are UTAP programming primitives enabling the interface to specify selections. The GET_SELECTION is a primitive to request a read of the configuration table, and subsequent mapping of a symbolic name into a system id number. The POST_SELECTION returns the id number. The message USE_SELECTION and appropriate numeric id designates the destination of the future messages. USE_SELECTION is a modal command and stays in effect until changed. USE_AXIS_MASK is a programming convenience for modules with multiple servos under coordinated control. The AXIS_MASK message contains a bitmask enumerating the selected axis.

The following code illustrates the use of the SELECTION message set. First, one does the symbolic mapping with GET_SELECTION to retrieve the appropriate ids for robot1 and robot2. (It is assumed that the Object Knowledge module responds with a POST_SELECTION, but is hidden in this example.) Then one alternates between robot selections to describe the coordinated motion. The addition of the BEGIN_BLOCK and END_BLOCK message primitives illustrates a mechanism to do concurrent motion control.

```
robot1 = GET_SELECTION("TLC:A:ROBOT:A");
robot2 = GET_SELECTION("TLC:A:ROBOT:B");
robot3 = GET_SELECTION("TLC:B:ROBOT:A");

BEGIN_BLOCK();
USE_SELECTION(robot1);
SET_POSITION(10.0,20.0,10.0,0.0, 90.0, 0.0);

USE_SELECTION(robot2);
SET_POSITION(20.0,20.0,10.0,0.0, 90.0, 0.0);
END_BLOCK();
```

The SELECTION message set is intended for multiple subordinates that are in need of programmatic control. It may be an option of the system to let the module itself designate the recipient of a set of messages. For example, suppose a system has a right and left arm that are functionally equivalent, then selection of the left or right subordinate can be context-driven or situation-dependent. If the superior is privy to some future knowledge that impacts the selection, it may select the best-fit. Otherwise, the subordinate could select a first-fit or random-fit.

10.3.4 Synchronization

Control systems require synchronization of devices within the system. For example, one would like for a tool change to complete before initiating the next tooling motion. Another possibility is that one desires that a series of motions be treated as one continuous motion without any interruption - or some dwell could occur.

Within the UTAP framework, synchronization is achieved with BEGIN_BLOCK and END_BLOCK generic messages. This construct is similar to the block concept in RS274. Messages that arrive between the BEGIN and END BLOCK messages are treated as a unit. It is assumed that the receiving module understands/describes how a set of operations are synchronized. The BLOCK set of messages is especially valuable for coordinating actions within the Parent Task Program Sequencer module in the UTAP.

The EVENT set of messages provides event-driven command sequencing. This is useful for operations that are sequenced to begin or end at some specified time within a BLOCK of commands, or begin or end when a concurrent operation reaches some state. The BEGIN_EVENT is embedded within a BLOCK construct and provides for an event to occur within the BLOCK either FROM_START, FROM_END, ERROR, or AT a specified time t. The ERROR event would call the currently loaded SAFE motion plan should some limit or error be encountered. (See following section for more details on this concept.) It is hoped to allow further exception handling and event triggered callbacks within the EVENT primitive at some future point in time.

10.4 Extensibility

The ability to broaden, shrink, or advance a system's functionality is required by the UTAP architecture and must be supported by the interfaces. This flexibility and extensibility must be achieved through explicit mechanisms. Extensibility within UTAP Interface Framework features 1) state context and parameter aggregating and naming, and 2) scalable and 3) seamless integration of sensors.

10.4.1 State Context Naming

The ability to aggregate and name a set of messages based on some state context information can greatly simplify programming. One of the small enhancements to the messaging system was the inclusion of the MACRO and PLAN set of messages.

The MACRO set of commands allow one to group and name a set of parameter messages. One sends a BEGIN_MACRO with a name, a set of parameter messages and then the END_MACRO message to define the macro. One uses the USE_MACRO command to invoke these parameter settings. Below is an example set of messages that may be grouped to describe a fragile or rigid parameter setting. Then, depending on the type of object, one selects the correct parameter context. For china, use delicate. For steel, use rigid.

```
US_BEGIN_MACRO (requires name, e.g., fragile or rigid)
    US_LOAD_TLC_DOF
    US_LOAD_TLC_FEED_RATE
    US_LOAD_TLC_TRAVERSE_RATE
    US_LOAD_TLC_CONTACT_FORCES
    US_LOAD_TLC_STIFFNESS_PARAMETERS
    US_LOAD_TLC_JOINT_GAIN_THRESHOLD
    US_LOAD_TLC_JOINT_SINGULARITY_THRESHOLD
END_MACRO

USE_MACRO fragile
```

Similarly, the PLAN set of commands allow one to aggregate and name a set of motions. This feature would be useful if one wanted to name a "SAFE" motion zone as a fallback motion in case of an error.

10.4.2 Scaling Control Dimensions

One peril involving generic interfaces is the requirement to satisfy a broad performance range. For example, some systems can provide a cost-effective solution with minimal complexity and were not intended for more sophisticated applications. One cannot require a simple but cost-effective module to accommodate the entire realm of interface possibilities. To be effective, the ability to scale the interface is needed.

Scaling within the Task Level Control module implies that its generic interface must accommodate simple position control as well as hybrid force-control with multi-device sensor fusion. Previous discussion illustrated the concept of scaling mode and parameter selection vectors. This section will discuss the use of option levels. The following message illustrates the levels one can have when defining the trajectory kinematic ring [BACKES2].

```
#define US_TLC_USE_KINEMATIC_RING_POSITIONING_MODE 626
struct us_tlc_use_kinematic_ring_msg_t {
        int id;
        Measurement_units_type units;
        enum {    BASE           = 0x0000001,
                  TOOL           = 0x0000002,
                  SENSOR_FUSION  = 0x0000004,
                  // RHS
                  DELTA          = 0x0000010,
                  OBJECT         = 0x0000020,
                  OBJECTBASE     = 0x0001000,
                  OBJECTOFFSET2  = 0x0002000,
                  OBJECTOFFSET3  = 0x0003000,
                  OBJECTOFFSET4  = 0x0004000,
           } ring_mask;
};
```

A selection mask is provided to allow the sophistication of the Task Level Control module to vary. The selection mask provides for one to define levels of positioning control - from simple position updates, to allowing sensor fusion and specifying transform models for the base, tool, object base. This range of specificity allows a broader range of modules to use the same generic interface without unnecessary expectations of a simpler control module.

10.4.3 Integration

An extension that one might desire is the ability to do on-line configuration and assignment of communications. For example, when you add a new sensor to your system, and you may wish to pump this data into an existing module. This can be especially difficult if real-time readings impact the control behavior. How do you accomplish this task? Generally, one would have a sensor fusion hook that allows sensors to pump readings into the sensor fusion slot.

One would desire the ability to say to a subordinate, "I want you to input readings from the so-and-so sensor and use this reading to calculate-an-offset/postmultiply/alter/delta-frame and modify the nominal goal action into an altered goal action." The use of the MACRO, GET message with the USE_SELECTION message enables this capability. Within the GET message, one specifies the routing information as a delta offset destination. Thus, the module uses a subordinate reading to modify its nominal goal with a delta offset. The following code demonstrates this concept. The code embeds a SELECTION and GET message within a MACRO message that will instruct a subordinate to use one of its subordinates readings as a delta offset value.

```
us_begin_macro("Sensor  Integration");
us_use_selection(us_get_selection("ROBOT.A:SENSOR.B"));
us_get_value(ROUTE._DELTA_OFFSET,
                         -1,                                // forever
                         .002,                              // every 2 milliseconds
                         0);                                // in delta offset location 1
us_end_macro("Sensor Integration");

us_use_selection(us_get_selection("ROBOT.A"));
us_use_macro("Sensor Fusion");
```

10.4.4 New Messages

The UTAP interfaces define a great many messages. Yet, it would be impossible to anticipate and explicitly enumerate every possible control and sensing algorithm and parameter. For example, suppose a better servo control algorithm is developed, how will the interface permit the selection of this algorithm? Further, suppose an additional compensation parameter could be specified within the servo control. How will the system adapt to the additional parametric capability?

Although it is hoped that the current set of messages do indeed satisfy the needs of the UTAP task scenarios, a mechanism is required for extending the scope of the interfaces. There exist many possibilities to extending the message set. Providing new definition construct (such as /'name' ... def in Postscript) within the interface language is one solution but greatly complicates the interface. For statically defined and published interfaces, the extensibility problem is inherent since the interface language does not provide general programming constructs. Without general programming constructs, supporting the requirements of portability and extensibility is difficult.

Adhering to a simple strategy for now, the current UTAP solution is to provide slot-holder messages to handle new messages. For comparison, the part description language RS274 provides macro slot holders, M80-M89, as a means of customization in a constrained manner. The UTAP extension will use the letters "EXT" to signify non-standard message extension. The extension messages include US_USE_EXT_ALGORITHM, US_LOAD_EXT_PARAMETER, US_SET_EXT_DATA_VALUE, US_GET_EXT_DATA_VALUE and US_POST_EXT_DATA_VALUE. Each message uses a numeric id to specify the extension instance. For example, id=1 specifies extension one, id=2 specifies extension two, etc.

The flexibility to add new messages comes at a cost. Complete interoperability can no longer be guaranteed. For example, RS274-defined parts that use the M80-89 macros contain implementation specific descriptions. These macros are rarely portable. UTAP extensibility will allow for technology to evolve, so that if an algorithm or feature extension becomes popular enough - so that more than one vendor is supporting it - then it can be transformed into a "regular" supported message in a later revision of the UTAP interfaces.

Table 1 – Module Classification

Module Class	Type	Examples
OI	POS Joystick Force Reflecting Joystick Pendant Panel Windows	Spaceball 6DOF F/R Joystick PIO: Screen, Buttons, Dials, Switches X-Windows, Windows
OM, OC, OK		
TD, TK, TDS	Teach Scripted Programmable	Record/Playback Motions Save/Load Parameters, Scripts Save/Load Program
TPS	Manipulation Navigation Tooling Machining Assembly	 (Not covered)
TPS: Transport	Teleop Guided Autonomous	Lift Tracked overhead gantry
TPS: Tooling	Contact Non-contact	Finishing Spraying
TPS: Machining	Horizontal	
TLC: AXIS_SERVO	SLM *SCARA *GANTRY *STEWART_PLATFORM	Serial Link Manipulator
TLC: SENSOR	FTS IMAGE PROBE *SWITCH	JR3 force torque sensors camera images LVDT beam-break, open/close gripper
TLC: TOOL	SPRAY FINISH *GRASP *SQUEEZE	coolant, paint, wash sand, grind, chamfer, debur - compliant grasp squeeze

Table 2 – Message Type Identification Table of Contents.

GENERIC:	101 - 199
ERROR:	-100 - -200
REMOTE	
AXIS_SERVO:	200 - 299
TOOL:	300 - 399
SENSOR:	400 - 499
PIO:	500 - 599
TLC:	600 - 699
DB:	700 - 799
VS:	800 - 899
LOCAL	
TDS:	1000 - 1099
TK:	1100 - 1199
TRD:	1200 - 1299
PTPS:	1300 - 1399
TPS:	1400 - 1499
OI:	1500 - 1599
OK:	1600 - 1699
SGD:	1700 - 1799
ADS:	1800 - 1899
SS:	2000 - 2099

Annex A
(informative)
Bibliography

[BACKES1] P. G. Backes, W. Zimmerman, M.B. Leahy, Jr., "Telerobotics Application to Aircraft Maintenance and Remanufacturing," IEEE International Conference on Robotics and Automation, San Diego, CA, 1994.

[BACKES2] P.G. Backes, "Dual-Arm Supervisory and Shared Control Space Servicing Task Experiments," AIAA Space Programs and Technologies Conference, Huntsville, AL, March 24-27, 1992.

[BOURNE] D.A. Bourne, D.T. Williams, "Satisfying the Needs of the Neutral Manufacturing Language with the Feature Exchange Language," Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 1990.

[EMC] F. M. Proctor, J.L. Michaloski, "Enhanced Machine Controller Architecture Overview," National Institute of Standards and Technology, Technical Report 5331, Gaithersburg, MD, December 1993.

[GISC] J.M. Greismeyer, M.J. McDonald, R.W. Harrigan, P.L Butler, J.B Rigdon, "Generic Intelligent System Control (GISC)," Sandia Internal Report, SAN92-2159, October 1992.

[KRAMER] T.R. Kramer, F.M. Proctor, J.M. Michaloski, "The NIST RS274/NGC Interpreter - Version 1," National Institute of Standards and Technology, Technical Report 5416, Gaithersburg, MD, April 1994.

[NASREM] J.S. Albus, H.G. McCain, R. Lumia, "NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM)" National Institute of Standards and Technology, Technical Report 1235, Gaithersburg, MD, 1989.

[ONIKA] D. B. Stewart, R.A. Volpe, P.K. Khosla, "Integration of Real-Time Software Modules for Reconfigurable Sensor-Based Control Systems," Proceedings of 1992 IEEE/RSI International Conference on Intelligent Robots and Systems (IROS '92), Raleigh, NC, July 7-10, 1992, pp. 325-332.

[RCS] J.S. Albus, "Outline for a Theory of Intelligence," IEEE Transactions on Systems, Man, and Cybernetics, 21(3), May/June 1991, pp. 473-509.

[SANCHO] M. McDonald, R.D. Palmquist, "Graphical Programming: On-Line Robot Simulation for Telerobotic Control," International Robots and Vision Automation Show and Conference Proceedings, April 5-8, 1993. Cobo Center, Detroit, Michigan USA, pp. 22-59 to 22-73.

[SMART] R. Anderson, "SMART: a modular architecture for robotics and teleoperation," Proceedings of the IEEE Conference on Robotics and Automation, Vol. 2, Atlanta, Georgia, May 1993, pp. 416-421.

[STA] R. Bajcsy, V. Kumar, M. Mintz, R. Paul, X. Yun, "A small-team architecture for multi-agent robotic systems," in Workshop on Intelligent Robotic Systems: Design and Applications, SPIE's Intelligent Robotics Symposium, (Boston, MA), November 1992.

[STELER] R.D. Steele, P.G. Backes, "ADA and Real-Time Robotics: Lessons Learned," IEEE Computer Magazine, Vol. 27(4), April, 1994, pp. 49-54.

[SUB] R. Brooks, "Elephants Don't Play Chess", Robotics and Autonomous Systems, No. 6, 1990, pp. 3-15.

[TCA] R. Simmons, "An architecture for coordinating, planning, sensing, and action," In Proceedings DARPA Planning Workshop, San Diego, CA, 1990, pp. 292-297.

[VCE] R.W. Harrigan, M.J. McDonald, B.R. Davies, "Remote Use of Distributed Robotics Resource to Enhance Technology Development and Insertion," ISRAM-94.

Annex B
(normative)
Component Analysis

The National Center for Manufacturing Sciences sponsorship of the Next Generation Controller project which defined the Specification for an Open System Architecture Standard (SOSAS) document [SOSAS]. This document provides an overview of philosophy and structure of the NGC. The SOSAS describes a reference architecture that is comprised of primitive components. From the reference architecture an application architecture is constructed that captures the functionality of the end systems at an abstract level. The selection of implementation components determines the final system.

Each component is an abstract encapsulation of funcionality

B.1 Application Architecture

AC 1 – Object Modeling:

RESPONSIBILITY: Object modeling provides for modeling of objects. This includes initial off-line description of objects and run-time model building.

INPUT: Object types, attributes, sensor data, operator input

OUTPUT: Updated object types, attributes, sensor data, operator input

AC 2 – Object Calibration:

RESPONSIBILITY: Calibration of an object's actual properties, e.g., position

INPUT: Object initial calibration properties

OUTPUT: Updated object calibration properties

AC 3 – Trajectory Description:

RESPONSIBILITY: Specify a trajectory for use in an application program

INPUT: Path information such as starting and end points, continuous inputs such as from a hand controller, preferred path segments

OUTPUT: Trajectory for insertion into task program

AC 4 – Object Knowledgebase:

RESPONSIBILITY: Store information about objects in the task environment including geometry and task information

INPUT: Object information

OUTPUT: Object information

AC 5 – Operator Input Devices:

RESPONSIBILITY: Transform operator input information into data for software modules and provide feedback to operator via input mechanisms

INPUT: Operator interaction and feedback data

OUTPUT: Operator input data to system modules, e.g., keyboard, audio, and handcontroller

AC 6 – Status & Graphical Display:

RESPONSIBILITY: Display system status including geometry, sensor data and task execution status

INPUT: Object status, task execution status, system status

OUTPUT: Status and geometry displays to operator, e.g., task execution status and geometric graphical display

AC 7 – Task Description & Supervision:

RESPONSIBILITY: Interactive task sequence (application program) generation and runtime interaction with operator for sequencing and sequence modification

INPUT: Information to generate application programs, e.g., task commands, macros, task sequences, object information, trajectories

OUTPUT: Application program including task programs

AC 8 – Analysis & Diagnosis:

RESPONSIBILITY: Analyze and diagnose task execution status

INPUT: Task execution status and task information on what and how to monitor

OUTPUT: Execution status and execution control commands such as stop

AC 9 – Subsystem Simulation:

RESPONSIBILITY: Provide a simulation of the task execution

INPUT: Task program commands

OUTPUT: Simulated subsystem control data

AC 10 – Task Knowledgebase:

RESPONSIBILITY: Provide task sequence building blocks such as macro commands and sequences

INPUT: Requests for commands or command types

OUTPUT: Macro commands and sequences with unbound parameters

AC 11 – Subsystem Task Program Sequencing:

RESPONSIBILITY: Sequence the subsystem task program commands

INPUT: Task programs and execution status

OUTPUT: Task level control commands and execution status

AC 12 – Parent Task Program Sequencing:

RESPONSIBILITY: Sequence the parent task program commands

INPUT: Parent task program and execution status

OUTPUT: Next command to execute

AC 13 – Subsystem Control:

RESPONSIBILITY: Execute task programs including closed loop and non-closed loop control

INPUT: Task program commands and command sequences and data from sensors, tools and mechanisms

OUTPUT: Status and commands to sensors, tools and mechanisms

AC 14 – Sensor Control:

RESPONSIBILITY: Provide control of a sensor

INPUT: Sensor commands and raw sensor data

OUTPUT: Processed sensor data and low level sensor commands

AC 15 – Sensing:

RESPONSIBILITY: Gather raw sensor data

INPUT: Low level sensor commands

OUTPUT: Raw sensor data

AC 16 – Tool Control:

RESPONSIBILITY: Control a tool

INPUT: Tool control commands and tool status

OUTPUT: Processed tool data and low level tool commands such as analog voltage

AC 17 – Tool Motion:

RESPONSIBILITY: Generate the tool motion such as by driving a motor

INPUT: Low level tool commands such as analog voltage

OUTPUT: Tool status

AC 18 – Robot Servo Control:

RESPONSIBILITY: Provide joint level servo control

INPUT: Robot configuration commands and status

OUTPUT: Joint status and low level robot joint commands such as voltage and status

AC 19 – Robot Motion:

RESPONSIBILITY: Generate the joint motion of the robot such as by driving a motor

INPUT: Low level robot joint commands

OUTPUT: Robot joint status

AC 20 – Virtual Sensor:

RESPONSIBILITY: Compute information as if it came from a real sensor, but using available data, such as for multiple sensor fusion

INPUT: Sensor commands and data

OUTPUT: Computed virtual sensor data

B.2 Hardware Architecture

HC 1 – Interface Controller:

RESPONSIBILITY: The interface controller is the computer which the operator uses to interact with the application programs. The supported interface may be simple, e.g., ascii inputs and outputs, or more complex, e.g., iconic interface with multiple input devices. The actual input devices such as voice input/output and hand controller, are treated as individual devices with their control programs.

INPUT: Inputs from operator input devices and status from task controller

OUTPUT: Status to operator, e.g., visual or audio, and task commands to task controller

HC 2 – Task Controller:

RESPONSIBILITY: The task controller is the physical computer hardware where the task level control of a task program is executed. The task sequencing software, task-device map, and device control software could also run on this controller.

INPUT: Task commands from the interface controller and status from the device controllers

OUTPUT: Commands to the device controllers and status to the interface controller

HC 3 – Device Controller:

RESPONSIBILITY: The device controller hardware receives the device commands from the task controller hardware. These commands will vary depending on the configuration (profile) of the controller. There are various options for the hardware profiles. The device controller could be a microprocessor card which has servo control software on it. It could take joint positions as input commands and output, to the device amplifier, analog or digital commands such as velocity or torque. Parameters for the servo control will also be communicated to the servo control. This same functionality could also be provided by buying a motion control card. For sensors, the device controller converts commands to low level sensor signals. When reading the sensor, the device controller processes the sensor data and provides it to the task controller. For tools, the device controller converts task controller tool commands to the low level device amplifier signals and returns to the task controller the tool status.

INPUT: Device commands from the task controller and status from the device amplifier

OUTPUT: Device amplifier signals to the the device amplifier and status to the task controller

HC 4 – Device Amplifier:

RESPONSIBILITY: The device amplifier provides the analog or digital control signal to the device. This could be analog voltage or a PWM signal to a motor drive or power to a sensor. A device amplifier module also generates the raw feedback data from the device.

INPUT: Device amplifier commands from the device controller and status from the device

OUTPUT: Control signals to the device and status to the device controller

HC 5 – Device:

RESPONSIBILITY: The device is the hardware that is being controlled or the sensor that is being used

INPUT: Control signals from the device amplifier

OUTPUT: Device action, e.g., motion or sensing

B.3 Software Components

The types of software modules in the system were described. Software modules in the system will now be described as software components with specified responsibilities, inputs and outputs. Some of the components could be further decomposed into multiple software modules. The application programs are treated here as software components and are given below.

B.3.1 System Software Components

SC 1 – Object Modeling:

RESPONSIBILITY: Provide functionality for modeling objects. This includes initial off-line description of objects and run-time model building.

INPUT: Object types, kinematics, attributes, operator input

OUTPUT: Updated object types, kinematics, attributes

SC 2 – Object Calibration:

RESPONSIBILITY: Calibration of an object's actual properties, e.g., position

INPUT: Object initial calibration properties, data on actual properties

OUTPUT: Updated object calibration properties

SC 3 – Trajectory Description:

RESPONSIBILITY: Generate a trajectory for use in an application program

INPUT: Path information such as starting and end points, continuous inputs such as from a hand controller, preferred path segments

OUTPUT: Trajectory for insertion into task program

SC 4 – Object Knowledgebase:

RESPONSIBILITY: Store information about objects in the task environment including geometry, task information and functions to call to acquire data

INPUT: Object information

OUTPUT: Object information

SC 5 – Operator Input Device Control:

RESPONSIBILITY: Transform operator input information into data for software modules and provide feedback through input devices, e.g., force reflection

INPUT: Operator inputs, e.g., keyboard, audio and handcontroller and feedback data

OUTPUT: Data to system modules and feedback to operator through input devices

SC 6 – Status & Graphical Display:

RESPONSIBILITY: Display system status including geometry, sensor data, and task execution

INPUT: Object status, task execution status, system status

OUTPUT: Status and geometry displays to operator

SC 7 – Task Program Editor:

RESPONSIBILITY: Provide an interactive interface to program developer for task sequence (application program) generation and modification

INPUT: Information to generate application programs, e.g., task commands, macros, task sequences, object information, trajectories, rules

OUTPUT: Application program including task programs

SC 8 – Task Program Supervisor:

RESPONSIBILITY: Provide operator based sequence execution control. Allows for single stepping commands, sending complete sequences and backing up. Allows task program editor to modify sequence.

INPUT: Application task programs, status, and operator inputs

OUTPUT: Task program commands or sequences to task program sequencers

SC 9 – Analysis & Diagnosis:

RESPONSIBILITY: Analyze and diagnose task execution status such as checking for collisions

INPUT: Task execution status and task information on what and how to monitor

OUTPUT: Execution status and execution control commands such as stop

SC 10 – Subsystem Simulation:

RESPONSIBILITY: Provide a simulation of the task execution with same inputs and outputs as the the real system

INPUT: Task program commands

OUTPUT: Simulated task level control system data

SC 11 – Task Knowledgebase:

RESPONSIBILITY: Provide task sequence building blocks such as macro commands and sequences

INPUT: Requests for commands or command types

OUTPUT: Macro commands and sequences with unbound parameters

SC 12 – Subsystem Task Program Sequencing:

RESPONSIBILITY: Sequence the subsystem task program commands

INPUT: Task programs and execution status

OUTPUT: Task level control commands and execution status

SC 13 – Parent Task Program Sequencing:

RESPONSIBILITY: Sequence the parent task program commands

INPUT: Parent task program and execution status

OUTPUT: Coordination commands to the subsystem task program sequencers

SC 14 – Subsystem Task Level Coordinated Control:

RESPONSIBILITY: Execute non-closed loop control of task programs

INPUT: Task program commands and data from sensors, tools and mechanisms

OUTPUT: Commands to task level closed loop control, sensors, tools and mechanisms and status

SC 15 – Subsystem Task Level Closed Loop Control:

RESPONSIBILITY: Execute closed loop control of task programs

INPUT: Commands to closed loop control modules and data from sensors, tools and mechanisms

OUTPUT: Commands to sensors, tools and mechanisms and status

SC 16 – Task Control Database:

RESPONSIBILITY: Store and provide information relevant to the task execution system such as status to be sent periodically to the object knowledgebase and actual data to be used which is associated with symbolic parameters

INPUT: Status data and database commands

OUTPUT: Status data

SC 17 – Sensor Control:

RESPONSIBILITY: Provide control of a sensor

INPUT: Sensor commands and raw sensor data

OUTPUT: Processed sensor data and low level sensor commands

SC 18 – Tool Control:

RESPONSIBILITY: Control a tool

INPUT: Tool control commands and tool status

OUTPUT: Low level tool commands such as analog voltage and tool status

SC 19 – Robot Servo Control:

RESPONSIBILITY: Provide joint level servo control

INPUT: Robot configuration commands and status

OUTPUT: Low level robot joint commands such as voltage and status

SC 20 – Virtual Sensor:

RESPONSIBILITY: Compute information as if it came from a real sensor, but using available data, such as for multiple sensor fusion

INPUT: Sensor commands and data

OUTPUT: Computed virtual sensor data

SC 21 – Motion Fusion:

RESPONSIBILITY: Combine the various sources of motion into a task level motion command for the mechanism

INPUT: Motion commands from the various motion control components, e.g., force control, visual servoing, trajectory generator, teleop motion; parameters specifying weights for each axis of control for each motion source

OUTPUT: A combined task level motion command for the mechanism. This is taken by the task-device map component and transformed into commands to the mechanism servoed axes.

SC 22 – Teleop Motion:

RESPONSIBILITY: Read hand controller inputs and generate motion commands for hand controller

INPUT: Hand controller motion input data from hand controller device control

OUTPUT: Motion command associated with teleoperation

SC 23 – Force Control:

RESPONSIBILITY: Perform closed loop force control and generate motion commands for force control

INPUT: Sensed force data, force setpoints, force control parameters

OUTPUT: Motion command associated with force control

SC 24 – Task Space Trajectory Generator:

RESPONSIBILITY: Generate task space pose trajectory and provide associated motion commands from trajectory generator

INPUT: Trajectory parameters, goal position, obstacle information

OUTPUT: Trajectory setpoints representing motion inputs from trajectory generator

SC 25 – Proximity Servo:

RESPONSIBILITY: Perform closed loop proximity control and generate motion commands for proximity control

INPUT: Sensed proximity data, proximity setpoints, proximity control parameters

OUTPUT: Motion command associated with proximity control

SC 26 – Orientation Servo:

RESPONSIBILITY: Perform closed loop orientation control and generate motion commands for orientation control

INPUT: Sensed orientation data, orientation setpoints, orientation control parameters

OUTPUT: Motion command associated with orientation control

B.3.1.1 Application Program Components

SC 27 – Mobile-Platform-Control:

RESPONSIBILITY: This program is for positioning the mobile platform in an appropriate location relative to the surface targeted for paint stripping. The mobile platform can be placed to the desired location and posture automatically by the system using the preassigned data or manually by the operator using the hand controller. The selection of automatic control mode in turn provides options for designating the platform location and posture. Manual control can share control with automatic control, as desired. The platform is fixed upon the completion of the task. A more advanced future system may have coordinated platform and manipulator control, but this would require accurate platform control and real-time platform-manipulator combination position information relative to the aircraft which will probably not be available in the first implementation.

INPUT: – PARAMETERS LIST –

– The final desired pose of the mobile platform in case of the automatic control mode

– The desired platform pose increment in case of the manual control mode

OUTPUT: – PARAMETERS LIST –

– The system state including current pose of the mobile platform

– The execution state, e.g., completion state for automatic control

SC 28 – Worksite-Registration:

RESPONSIBILITY: This program is for configuring the manipulator in an appropriate position of the worksite to start the desired tool (gun) motion. The position of the manipulator base relative to the aircraft skin area to be stripped is determined. The data required for the worksite registration should be provided as input, which include the distance of tool separation, the orientation of tool with respect to the surface, and the landmarks or features of targeted surface (such as the surface curvature that can be measured by the force/torque or tactile sensor). The manual mode of control combined with the sensor-based distance and orientation servos can accomplish this task. This program and the mobile-platform-control can be used together for two consecutive paint-stripping operations.

INPUT: – PARAMETERS LIST –

– The data for worksite registration, including the stand-off distance and the orientation of the tool with respect to the targeted surface

– The priority setting between manual and sensor-based automatic operation

OUTPUT: – PARAMETERS LIST –

– The measured data from the sensors

– The current manipulator pose

SC 29 – Paint-Stripping-Operation:

RESPONSIBILITY: This program is for stripping paint off the skin of a large aircraft based on supervisory and shared telerobotics control. Prior to running this program, the following initialization is required:

- Setting up the mobile platform to an appropriate position.
- Configuring the manipulator at the initial pose for worksite registration.

Upon execution of this program the operator is asked whether the initial set-up by moving the mobile platform to the desired location and configuring the manipulator at the start position are done. If not, the operator should open the mobile-platform-control and worksite-registration programs by clicking the corresponding icons and execute them for the initialization. Once the initial set-up is completed, the operator is asked to assign the system parameters and control modes.

INPUT: – PARAMETERS LIST –

- Off line generated tool path
- Run time generated tool path
- The desired separation/stand-off of the tool from the target surface
- The desired tool orientation relative to the target surface
- The constraints on tool path such as the constrained motion imposed on individual axes
- Tool speed
- PMB (Plastic Media Bead) pressure
- The selection of the motion input sources affecting the tool (which will indicate the desired control mode)

OUTPUT: – PARAMETERS LIST –

- Commands to the task control including equivalent commands associated with the application command inputs above
- Status of execution including manipulator status, sensors' status, and current subtask status for use in analysis and display to the operator

Annex C
(normative)

Environment Profile Suite

Annex C contains a list of the profiles that can be used to generate a UTAP system specification. Each module in the system would be required to fill out one, maybe several, generic, error and data knowledge profiles - depending on the number of upper and subsystem links in the system. For now, only the local modules have been specified.

C.1 Application Environment Profile

A UTAP module *shall* conform to the environment which includes *system profile* that names each hardware device and device profile in the system. A device could be a computer or control device. The system profile runs under a *system environment* which is also profiled. This system environment profile

Computer boards have a device profile that includes CPU type, CPU characteristics and the CPU performance characteristics. Included profile is the operating system support for the CPU.

Controller boards are devices that would have a application-specific profile.

The system environment describes the infrastructure support (such as communication mechanisms) and resources (disks, extra memory, etc.) available to system devices.

Table C.1 – System Profile

DEVICE	Device Profile	Platform
...	...	...

Table C.2 – System Environment Profile

Bus:	________________
Memory:	________________
Disk:	________________
Disk Memory:	________________
Floppy Disk:	________________
Floppy Size:	________________
Floppy Density:	________________
Compact Disk :	________________
LAN Cabling:	________________
LAN Protocol:	________________

Table C.3 – Processor Board Profile

Board Id	
CPU Type	
Memory:	
OS Type:	
OS Version:	
OS Release:	
BUS Support	
IO Support	
Peripheral Support	
Special:	

C.2 Interface Environment Profile

A universal format is assumed in developing the message interfaces. It is assumed that each module displays a similar flow of messages. Figure C.1 illustrates the message flow of a module. A message naming convention is used for improved comprehension. Within a messages, a keyword is used to delineate between mode/goal/state information. The keywords are grouped by type:

- MESSAGING (i.e., BLOCK, MACRO, PLAN, EVENT, SELECTION)

- SEQUENCING CONTROL: generics (i.e., STARTUP, SHUTDOWN, ENABLE, DISABLE, etc.)

- MODALITY: USE, START, STOP, COMPUTE

- PARAMETRIC: LOAD, INCREMENT, SELECT

- DATA COMMAND: SET, ADJUST, GET

- STATUS: POST

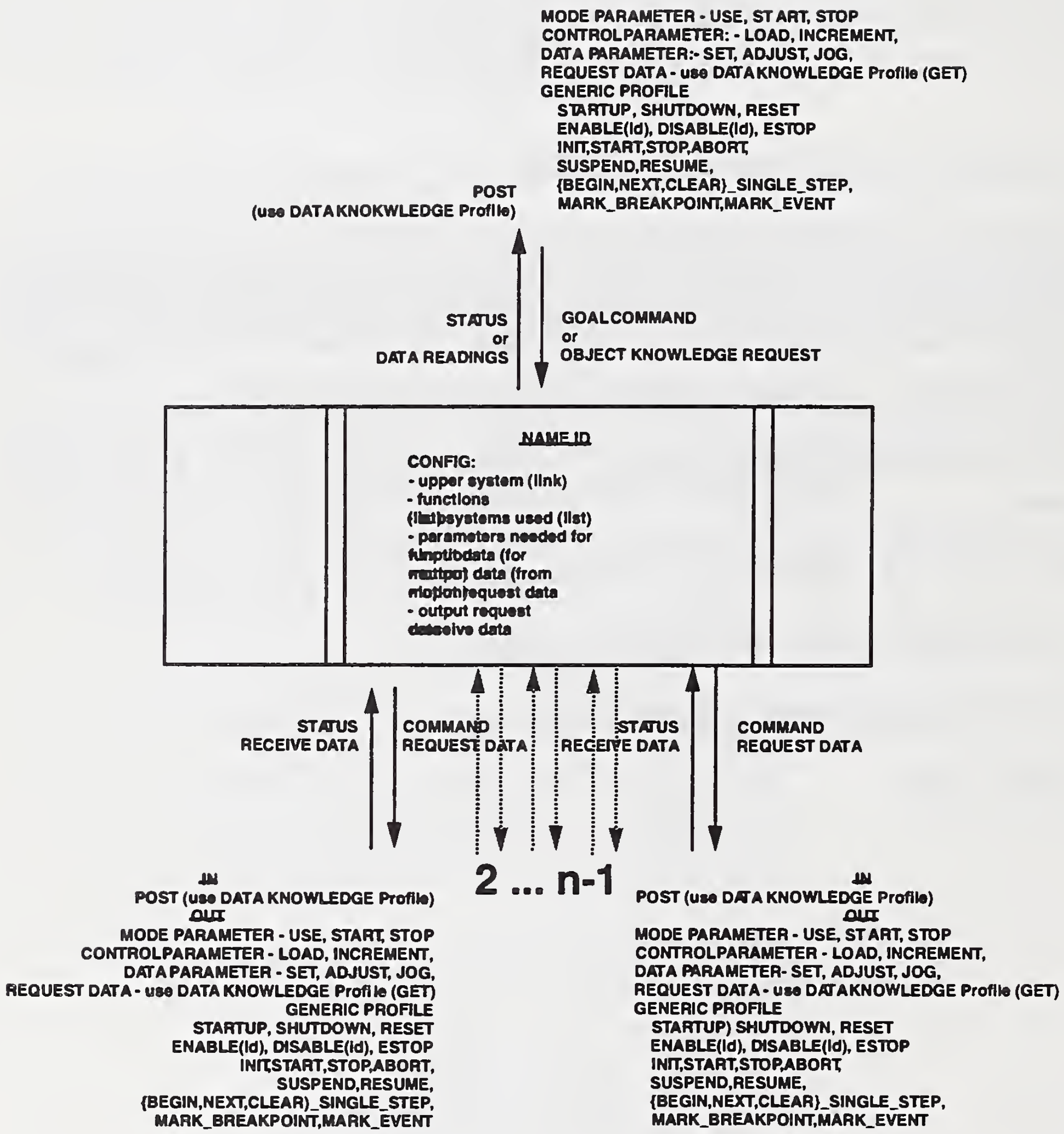
MODE PARAMETER - USE, START, STOP
CONTROLPARAMETER: - LOAD, INCREMENT,
DATA PARAMETER:- SET, ADJUST, JOG,
REQUEST DATA - use DATAKNOWLEDGE Profile (GET)
GENERIC PROFILE
STARTUP, SHUTDOWN, RESET
ENABLE(Id), DISABLE(Id), ESTOP
INIT,START,STOP,ABORT,
SUSPEND,RESUME,
{BEGIN,NEXT,CLEAR}_SINGLE_STEP,
MARK_BREAKPOINT,MARK_EVENT
POST
(use DATAKNOKWLEDGE Profile)
STATUS
or
DATA READINGS
GOALCOMMAND
or
OBJECT KNOWLEDGE REQUEST
NAME ID
CONFIG:
- upper system (link)
- functions
(list)bsystems used (list)
- parameters needed for
funptibdata (for
mattpo) data (from
riopioh)equest data
- output request
dateeive data
STATUS
RECEIVE DATA
COMMAND
REQUEST DATA
STATUS
RECEIVE DATA
COMMAND
REQUEST DATA
2 ... n-1
IN
POST (use DATA KNOWLEDGE Profile)
OUT
MODE PARAMETER - USE, START, STOP
CONTROLPARAMETER - LOAD, INCREMENT,
DATAPARAMETER - SET, ADJUST, JOG,
REQUEST DATA - use DATA KNOWLEDGE Profile (GET)
GENERIC PROFILE
STARTUP, SHUTDOWN, RESET
ENABLE(Id), DISABLE(Id), ESTOP
INIT,START,STOP,ABORT,
SUSPEND,RESUME,
{BEGIN,NEXT,CLEAR)_SINGLE_STEP,
MARK_BREAKPOINT,MARK_EVENT
IN
POST (use DATA KNOWLEDGE Profile)
OUT
MODE PARAMETER - USE, START, STOP
CONTROLPARAMETER - LOAD, INCREMENT,
DATA PARAMETER- SET, ADJUST, JOG,
REQUEST DATA - use DATAKNOWLEDGE Profile (GET)
GENERIC PROFILE
STARTUP) SHUTDOWN, RESET
ENABLE(Id), DISABLE(Id), ESTOP
INIT,START,STOP,ABORT,
SUSPEND,RESUME,
{BEGIN,NEXT,CLEAR}_SINGLE_STEP,
MARK_BREAKPOINT,MARK_EVENT

Figure C.1 – Module Profile Specification

Table C.4 – Generic Message Profile

	Configuration
Name	______
Module Type	______

	MESSAGE PRIMITIVES	*Comply*
	BLOCK	___
	MACRO	___
	PLAN	___
	EVENT	___
	USE_SELECTION	___

	SEQUENCE	*Comply*
Powerup	STARTUP	___
	SHUTDOWN	___
	RESET	___
S/E	ENABLE	___
	DISABLE	___
	ESTOP	___
Software	START	___
	STOP	___
	HALT	___
	HOLD	___
	SUSPEND	___
	RESUME	___
	BEGIN_SINGLE_STEP	___
	NEXT_SINGLE_STEP	___
	CLEAR_SINGLE_STEP	___
	MARK_BREAKPOINT	___
	MARK_EVENT	___
Status	LOAD_STATUS_TYPE	___
	LOAD_STATUS_PERIOD	___
	STATUS_REPORT	___

	OBJECT DATA	*Comply*
	POST_ID	___
	GET_OBJECT_ID	___
	USE_OBJECT	___
	GET_FEATURE	___
	USE_FEATURE	___
	GET_VALUE	___
	POST_VALUE	___
	GET_LIST	___
	POST_LIST	___

Table C.5 – Data Knowledge

	Configuration
Name	______
Module Type	______

ATTRIBUTES	*Actual*	*Desired*	*Max*	*Min*	*Avg*	*History*
POST_RESPONSE	___	___	___	___	___	___
GET_SELECTION	___	___	___	___	___	___
GET_TIME	___	___	___	___	___	___
GET_POSITION	___	___	___	___	___	___
GET_ORIENTATION	___	___	___	___	___	___
GET_POSE	___	___	___	___	___	___
GET_VELOCITY	___	___	___	___	___	___
GET_ACCELERATION	___	___	___	___	___	___
GET_JERK	___	___	___	___	___	___
GET_FORCE	___	___	___	___	___	___
GET_TORQUE	___	___	___	___	___	___
GET_MASS	___	___	___	___	___	___
GET_TEMPERATURE	___	___	___	___	___	___
GET_PRESSURE	___	___	___	___	___	___
GET_VISCOSITY	___	___	___	___	___	___
GET_LUMINANCE	___	___	___	___	___	___
GET_HUMIDITY	___	___	___	___	___	___
GET_GEOMETRY	___	___	___	___	___	___
GET_TOPOLOGY	___	___	___	___	___	___
GET_SHAPE	___	___	___	___	___	___
GET_PATTERN	___	___	___	___	___	___
GET_MATERIAL	___	___	___	___	___	___
GET_KINEMATICS	___	___	___	___	___	___

Table C.6 – Errors

	Configuration
Name	______
Module Type	______

ERRORS	*Comply*
POSIX ERRORS	___
CMD_NOT_IMPLEMENTED	___
ERROR_COMMAND_ENTRY	___
ERROR_DUPLICATE_NAME	___
ERROR_BAD_DATA	___
ERROR_NO_DATA_AVAIL	___
SAFETY_VIOLATION	___
LIMIT_EXCEEDED	___
ERROR_OVER_SPECIFIED	___
ERROR_UNDER_SPECIFIED	___

Table C.7 – Axis Servo Command Profile

	Configuration	*Comply*
Name	________	
Module Type	US_AXIS_SERVO	
Links_Up:	TASK_LEVEL_CONTROL	
Links_Down:	PIO	
Other:	________________	
	________________	
Interface:	GENERICS	________________
	DATA KNOWLEDGE	________________
	ERRORS	________________
	MODES	*Comply*
Safety:	SET_BRAKES	____
	CLEAR_BRAKES	____
	MODES	*Comply*
	INPUT	
Units:	USE_ANGLES	____
	USE_RADIANS	____
Reference:	USE_ABSOLUTE	____
	USE_RELATIVE	____
Representation:	USE_POSITION	
	USE_VELOCITY	____
	USE_CURRENT	____
	USE_VOLTAGE	____
	USE_FEEDFORWARD_TORQUE	____
	SELECTIONS	*Comply*
Compensation:	USE_PID	____
	USE_STIFFNESS	____
	USE_IMPEDANCE	____
	USE_COMPLIANCE	____
	AUGMENTATIONS	
	{START—STOP}_GRAVITY_COMP	
	PARAMETER LOADS	*Comply*
	LOAD_DOF	____
	LOAD_CYCLE_TIME	____
	LOAD_AXIS_MASK	____
	LOAD_STATUS_UPDATE	____
	LOAD_SAMPLING_PERIOD	____
	LOAD_FREQUENCY_RESPONSE	____
Gains:	LOAD_PID	____
	LOAD_DAMPING_VALUES	____
Limits:	LOAD_JOINT_LIMIT	____
	LOAD_VELOCITY_LIMIT	____
	LOAD_GAIN_LIMIT	____

Table C.8 – Axis Servo Data Profile

Configuration

	Input Command Data	SET	ADJ	JOG
	POSITION	___	___	___
	VELOCITY	___	___	___
	TORQUE	___	___	___
	VELOCITY	___	___	___
	ACCELERATION	___	___	___
	FORCES	___	___	___

	Request Input/ Output Posted	SET	ADJ	JOG
Position:	ACTUAL	___	___	___
	DESIRED	___	___	___
	MAX	___	___	___
	MIN	___	___	___
	HISTORY(-t,t0)	___	___	___
Velocity:	ACTUAL	___	___	___
	DESIRED	___	___	___
	MAX	___	___	___
	MIN	___	___	___
	HISTORY(-t,t0)	___	___	___

Output Status
See – Data Knowledge

SUBSYSTEM LINK

Name: ________

Command Data
See – Subsystem Module Profile ________

Request Data
See – Data Knowledge ________

Receive Data
See – Data Knowledge ________

Table C.9 – Tool Control Profile - Spindle

	Configuration	*Comply*
Name	________	
Module Type	US_TOOL_	
Class:	"Spindle"	
Links_Up:	TASK_LEVEL_CONTROL	
Links_Down:	PIO	
Other:	________________	
Interface:	GENERICS	____
	DATA KNOWLEDGE	____
	ERRORS	____
	MODES	*Comply*
	SELECTIONS	
	START TURNING	____
	STOP TURNING	____
	LOCK_Z	____
	UNLOCK_Z	____
	AUGMENTATIONS	
	USE_FORCE	____
	USE_NO_FORCE	____
	PARAMETER LOADS	*Comply*
	LOAD_SPEED	____
	SPINDLE_ORIENT	____
	Input Data Accepted	*Comply*
	SPINDLE_RETRACT_TRAVERSE	____
	SPINDLE_RETRACT	____
	SPINDLE_ORIENT	____
	Output Status	
	See – Data Knowledge	
	SUBSYSTEM LINK	
Name:	________	
	Command Data	
	See – Subsystem Module Profile	________
	Request Data	
	See – Data Knowledge	________
	Receive Data	
	See – Data Knowledge	________

Table C.10 – Tool Control Profile - Coolant

	Configuration	*Comply*
Name	________	
Module Type	US_TOOL_	
Class:	"Spindle"	
Links_Up:	TASK_LEVEL_CONTROL	
Links_Down:	PIO	
Other:	________________	
Interface:	GENERICS	____
	DATA KNOWLEDGE	____
	ERRORS	____
	MODES	*Comply*
	SELECTIONS	
	START MIST	____
	STOP_MIST	____
	START_FLOOD	____
	STOP_FLOOD	____
	Input Data Accepted	
	See – Data Knowledge	
	Output Status	
	See – Data Knowledge	
	SUBSYSTEM LINK	
Name:	________	
	Command Data	
	See – Subsystem Module Profile	________
	Request Data	
	See – Data Knowledge	________
	Receive Data	
	See – Data Knowledge	________

Table C.11 – Generic Sensor

	Configuration	*Comply*
Name	________	
Module Type	US_SENSOR_	
Class:	"Generic"	
Links_Up:	TASK_LEVEL_CONTROL	
Links_Down:	PIO	
Other:	________________	
Interface:	GENERICS	____
	DATA KNOWLEDGE	____
	ERRORS	____
	MODES	*Comply*
	use: USE_MEASUREMENT_UNITS	
	SELECTIONS	
	START/STOP US_START_TRANSFORM	
	US_STOP_TRANSFORM	____
	US_START_FILTER	____
	US_STOP_FILTER	____
	PARAMETER	
load:	US_LOAD_SAMPLING_SPEED	____
	US_LOAD_FREQUENCY	____
	US_LOAD_TRANSFORM	____
	US_LOAD_FILTER	____
	Input Data Accepted	
set:	POSITION	____
	ORIENTATION	____
	Output Status Posted	*Comply*
	See Data Knowledge	________
post:	SENSOR_POST_READING	
	SCALAR_SENSOR_POST_READING	
	VECTOR_SENSOR_POST_READING	
	Output Status	
	See – Data Knowledge	
	SUBSYSTEM LINK	
Name:	________	
	Command Data	
	See – Subsystem Module Profile	________
	Request Data	
	See – Data Knowledge	________
	Receive Data	
	See – Data Knowledge	________
get:	GET_READING	
	GET_ATTRIBUTES_READING	

Table C.12 – Sensor - Image

	Configuration	*Comply*
Name	________	
Module Type	US_SENSOR_	
Class:	"Image"	
Links_Up:	TASK_LEVEL_CONTROL	
Links_Down:	PIO	
Other:	________________	
Interface:	GENERICS	____
	DATA KNOWLEDGE	____
	ERRORS	____
	MODES	*Comply*
use:	US_IMAGE_USE_FRAME_GRAB_MODE	____
	US_IMAGE_USE_HISTOGRAM_MODE	____
	US_IMAGE_USE_CENTROID_MODE	____
	US_IMAGE_USE_GRAY_LEVEL_MODE	____
	US_IMAGE_USE_THRESHOLD_MODE	____
	US_IMAGE_COMPUTE_SPATIAL_DERIVATIVES_MODE	____
	US_IMAGE_COMPUTE_TEMPORAL_DERIVATIVES_MODE	____
	US_IMAGE_USE_SEGMENTATATION_MODE	____
	US_IMAGE_USE_RECOGNITION_MODE	____
	US_IMAGE_COMPUTE_RANGE_MODE	____
	US_IMAGE_COMPUTE_FLOW_MODE	____
	PARAMETER	*Comply*
load:	US_IMAGE_LOAD_CALIBRATION	____
	Input Data Accepted	
set:	POSITION	____
	ORIENTATION	____
	US_IMAGE_ADJUST_POSITION	____
	US_IMAGE_ADJUST_FOCUS	____
	Output Status Posted	
	See – Data Knowledge	
post:	US_2D_SENSOR_POST_READING	____
	US_IMAGE_POST_SPECIFICATION	____
	US_IMAGE_POST_PIXEL_MAP_READING	____
	US_IMAGE_POST_HISTOGRAM_READING	____
	US_IMAGE_POST_XY_CHAR_READING	____
	US_IMAGE_POST_BYTE_SYMBOLIC_READING	____
	US_IMAGE_POST_THRESHOLD_READING	____
	US_IMAGE_POST_SPATIAL_DERIVATIVE_READING	____
	US_IMAGE_POST_TEMPORAL_DERIVATIVE_READING	____
	US_IMAGE_POST_RECOGNITION_READING	____
	US_IMAGE_POST_RANGE_READING	____
	US_IMAGE_POST_FLOW_READING	____
	Request Data	
	– See Data Knowledge ________	
get:	GET_READING	____
	GET_ATTRIBUTES_READING	____
	Receive Data	
	See – Data Knowledge ________	

Table C.13 – Subsystem Task Level Control

	Configuration	*Comply*
Name	________	
Module Type	US_TLC_	
Class:	"Generic"	
Links_Up:	TASK_PROGRAM_SEQUENCER	
Links_Down:	________	
Other:	________________	
Interface:	GENERICS	____
	DATA KNOWLEDGE	____
	ERRORS	____
	MODES	*Comply*
use:	US_TLC_USE_JOINT_REFERENCE_FRAME	____
	US_TLC_USE_CARTESIAN_REFERENCE_FRAME	____
	US_TLC_USE_REPRESENTATION_UNITS	____
	US_TLC_USE_ABSOLUTE_POSITIONING_MODE	____
	US_TLC_USE_RELATIVE_POSITIONING_MODE	____
	US_TLC_USE_WRIST_COORDINATE_FRAME	____
	US_TLC_USE_TOOL_TIP_COORDINATE_FRAME	____
	US_TLC_USE_MODIFIED_TOOL_LENGTH_OFFSETS	____
	US_TLC_USE_NORMAL_TOOL_LENGTH_OFFSETS	____
	US_TLC_USE_NO_TOOL_LENGTH_OFFSETS	____
	US_TLC_USE_KINEMATIC_RING_POSITIONING_MODE	____
load:	US_TLC_LOAD_DOF	____
	US_TLC_LOAD_CYCLE_TIME	____
	US_TLC_LOAD_REPRESENTATION_UNITS	____
	US_TLC_LOAD_LENGTH_UNITS	____
	US_TLC_LOAD_RELATIVE_POSITIONING	____
	US_TLC_ZERO_RELATIVE_POSITIONING	____
	US_TLC_ZERO_PROGRAM_ORIGIN	____
	US_TLC_LOAD_KINEMATIC_RING_POSITIONING_MODE	____
	US_TLC_LOAD_BASE_PARAMETERS	____
	US_TLC_LOAD_TOOL_PARAMETERS	____
	US_TLC_LOAD_OBJECT	____
	US_TLC_LOAD_OBJECT_BASE	____
	US_TLC_LOAD_OBJECT_OFFSET	____
	US_TLC_LOAD_DELTA	____
	US_TLC_LOAD_OBSTACLE_VOLUME	____
	US_TLC_LOAD_NEIGHBORHOOD	____
	US_TLC_LOAD_FEED_RATE	____
	US_TLC_LOAD_TRAVERSE_RATE	____
	US_TLC_LOAD_ACCELERATION	____
	US_TLC_LOAD_JERK	____
	US_TLC_LOAD_PROXIMITY	____
	US_TLC_LOAD_CONTACT_FORCES	____
	US_TLC_LOAD_JOINT_LIMIT	____
	US_TLC_LOAD_CONTACT_FORCE_LIMIT	____
	US_TLC_LOAD_CONTACT_TORQUE_LIMIT	____
	US_TLC_LOAD_SENSOR_FUSION_POS_LIMIT	____
	US_TLC_LOAD_SENSOR_FUSION_ORIENT_LIMIT	____
	US_TLC_LOAD_SEGMENT_TIME	____
	US_TLC_LOAD_TERMINATION_CONDITION	____
	US_TLC_INCR_VELOCITY	____
	US_TLC_INCR_ACCELERATION	

Table C.14 – Subsystem Task Level Control - cont.

selections:	US_TLC_START_MANUAL_MOTION	____
	US_TLC_STOP_MANUAL_MOTION	____
	US_TLC_START_AUTOMATIC_MOTION	____
	US_TLC_STOP_AUTOMATIC_MOTION	____
	US_TLC_START_TRAVERSE_MOTION	____
	US_TLC_STOP_TRAVERSE_MOTION	____
	US_TLC_START_GUARDED_MOTION	____
	US_TLC_STOP_GUARDED_MOTION	____
	US_TLC_START_COMPLIANT_MOTION	____
	US_TLC_STOP_COMPLIANT_MOTION	____
	US_TLC_START_FINE_MOTION	____
	US_TLC_STOP_FINE_MOTION	____
	US_TLC_START_MOVE_UNTIL_MOTION	____
	US_TLC_STOP_MOVE_UNTIL_MOTION	____
	US_TLC_START_STANDOFF_DISTANCE	____
	US_TLC_STOP_STANDOFF_DISTANCE	____
	US_TLC_START_FORCE_POSITIONING_MODE	____
	US_TLC_STOP_FORCE_POSITIONING_MODE	____
	Input Data Accepted	
	US_TLC_SET_GOAL_POSITION	____
	US_TLC_GOAL_SEGMENT	____
	US_TLC_ADJUST_AXIS	
	Output Status	
	See – Data Knowledge	
	SUBSYSTEM LINK	
Name:	________	
	Command Data	
	See – Subsystem Module Profile	________
	Request Data	
	See – Data Knowledge	________
	Receive Data	
	See – Data Knowledge	________
	US_TLC_UPDATE_SENSOR_FUSION	____

Table C.15 – Subsystem Task Level Control

	Configuration	*Comply*
Name	________	
Module Type	US_TLC_	
Class:	"Generic"	
Links_Up:	TASK_PROGRAM_SEQUENCER	
Links_Down:	____	
Other:	________________	
Interface:	GENERICS	____
	DATA KNOWLEDGE	____
	ERRORS	____
	MODES	*Comply*
	Task Level Generics	____
use:	US_TLC_SELECT_PLANE	____
	US_TLC_USE_CUTTER_RADIUS_COMPENSATION	____
load:	US_TLC_LOAD_DOF	____
selections:	US_TLC_START_CUTTER_RADIUS_COMPENSATION	____
	US_TLC_STOP_CUTTER_RADIUS_COMPENSATION	____
	Input Data Accepted	
	US_TLC_STRAIGHT_TRAVERSE	____
	US_TLC_ARC_FEED	____
	US_TLC_STRAIGHT_FEED	____
	US_TLC_PARAMETRIC_2D_CURVE_FEED	____
	US_TLC_PARAMETRIC_3D_CURVE_FEED	____
	US_TLC_NURBS_KNOT_VECTOR	____
	US_TLC_NURBS_CONTROL_POINT	____
	US_TLC_NURBS_FEED	
	Output Status	
	See – Data Knowledge	
	SUBSYSTEM LINK	
Name:	________	
	Command Data	
	See – Subsystem Module Profile	________
	Request Data	
	See – Data Knowledge	________
	Receive Data	
	See – Data Knowledge	________
	US_TLC_UPDATE_SENSOR_FUSION	____

Annex D
(informative)
Examples

D.1 API Interface Example

The UTAP message format provides the size and structure for the interfaces. The UTAP messages define the information that crosses the communication channel (or link or wire). This message interface is supposed to be flexible, but not necessarily suitable for application programming. One may require an API to sit between the message interface and the programmer, much like a device driver hides the implementation details of a device. Figure D.1 illustrates the relationship between the superior and the subordinate in such a setup. An API exists in the superior as a abstraction mechanism for communicating with the subordinate.

The programmer can use the UTAP isomorphic functional API or can use existing software that has a customized middleware to map the application code into the UTAP message interface. This section will present an example that illustrates the first possibility - using the UTAP isomorphic functional API for servo control. The hope is that this functional API is similar to most existing products and can be achieved by renaming keywords with new procedural names and reordering the procedural parameters.

An example to control a 1DOF servo from the task level control module will be developed. In this example, the first point of illustration will be to use the API to define a hi-gain and a low-gain mode. Within the example, the API subroutine calls still use the heap (or pointer to the data) concept to pass parameters.

```
hi_gain(){
    double p=100, i=200, d=20;
    double ilimit=30;
        us_begin_macro("hi_gain");
        us_axis_servo_load_pid_gain(&p,&i,&d);
        limiti=250; us_axis_servo_load_integration_limit(&ilimit);
        us_end_macro();
}

lo_gain(){
    double p=50, i=200, d=20;
    double ilimit=30;
        us_begin_macro("lo_gain");
        us_axis_servo_load_pid_gain(&p,&i,&d);
        ilimit=250; us_axis_servo_load_integration_limit(&ilimit);
        us_end_macro();
```

Once we have the parameters and modes defined, we can then work on the process model. Within the SERVO interface process, you would need to startup, run, and shutdown.

```
Servo_Interface_init() {
        us_axis_servo_load_dof(1);          // assign dof for heap mgt.
        us_axis_servo_use_degree_units();   // prefer degrees to radians
        us_servo_init();                    // init servos
        us_servo_enable();                  //
        hi_gain();                          // set up hi gain
        lo_gain();                          // set up lo gain
    }
```

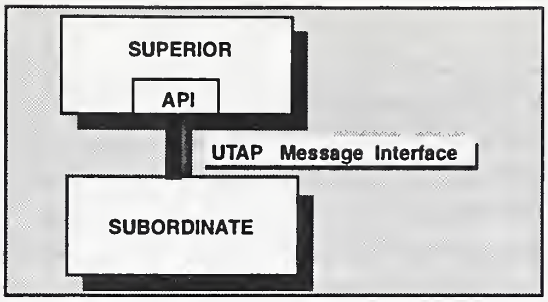

Figure D.1 – Superior use of API Interface to Command Subordinate

The actual process will be to initialize the servo, use hi-gain parameters with PID control mode, and then move to a desired joint position. The concept of getting and updating readings of the actual position will also be developed. The test fuzzy_equal was coined to provide an approximately equal function.

```
Servo_Interface() {
        double joint;
        us_use_selection(us_get_selection("DOFSERVO1"));
        us_axis_servo_load_absolute_positioning();
        us_startup();
        Servo_interface_init();
        us_use_macro("hi_gain");
        us_use_pid();
        us_axis_servo_home    ();                        // reset servos
        us_start();                                      // now the system will move home

        // One time move
        joint = 10;
        us_axis_servo_set_position(joint);

        // Post actual readings
    { ROUTE p;
        double now;
            p.type = ROUTE.STATUS;
            do { now=us_axis_servo_get_position(p , Modifier_t._real);
                while(!fuzzy_equal(now, joint));
            }
    }
    us_axis_servo_disable();
    us_axis_servo_shutdown();
}
```

This example illustrated a simple servo interface. Although illustrative it presented an ad hoc solution. One would prefer to use a more elegant internal control architecture (e.g., see [RCS], [ONIKA], [GISC], [STELER], [TCA], among others cited in Bibliography) so that one has better coordination of the sensing, world modeling and behavior generation aspects of intelligent control.

D.1.1 Tool Manipulation

One can program the tool in several methods. A superior module can enable the tool in the kinematic ring bitmask, and then send the tool transform. Another option is for the interface to use CHANGE_TOOL and TOOL_OFFSET messages and override the kinematic ring selection mask.

The tool offset messages are more in line with traditional NC tool programming (see [KRAMER]). Within the UTAP interface, it will be assumed that there is a table of 40 tool length offset numbers, and one or more registers, each with a tool length offset modifier number. There is a tool length offset mode which can be set to one of three values: NONE, NORMAL, and MODIFIED. In the NONE mode, no tool length offsets are used. In the NORMAL mode, the tool length offset value in the position of the table with the same index as the tool currently in the spindle is used. In the MODIFIED mode, the value used for the tool length offset is the modifier number in the currently selected modifier register added to the offset value for the tool currently in the spindle. There are currently no commands for setting the values in the tool length offset table or for setting the values of the modifier numbers in the modifier registers.

D.1.2 Sensor Programming Example

The sensor messages were categorized by dimensionality. The sensors were generically grouped as scalar, vector, and 2D array. Across each category, the GET_READINGS message is generally universal. On the other hand, posting messages were customized according to expected sensors readings. For example, although one can construct a force/torque query message from generic building blocks, it is redundant since this sensor is so common. (For example one can use the generic message GET_DATA_LIST with attribute = _force | _torque.) Wherever possible, sensor readings that were anticipated to be common were given a distinguishing message name.

The following example outlines an interface to a force torque sensor.

```
ROUTE route;
Attribute_t attr;
Modifier_t modifier;
us_ft_sensor_post_reading_t reading;
double fx,fy,fz;                                            // force
double tx,ty,tz;                                            // torque

    // setup parameter attribute and modifier info
    attr = Attribute_t.force | Attribute_t.torque;
    modifier=Modifier_t.actual;

    // setup routing info
    route.type = ROUTE.STATUS;
    route.times = 1;

    us_use_selection(us_get_selection("TLC:A:FORCE_TORQUE_SENSOR"));
    us_load_dof(3);
    us_load_sampling_speed(100Hz);                          // in Hertz
    us_load_frequency(.10);                                 // update every 100 milliseconds
    us_load_filter(us_sensor_load_filter_msg_t.HI_PASS,1000);
    us_start_filter();                                      // start filter
    us_start();

    //
    while(1)
    { reading = us_sensor_get_attributes_reading(route,attr,modifier);
      fx=reading[0]; fy= reading[1]; fz=reading[2];
      tx=reading[3]; ty= reading[4]; tz=reading[5];
      // Now, do something with the values....
    }
```

D.2 Channel Interface Example

As suggested, the communication protocol is treated as a separate issue from language or messaging strategy. Just like the C language file descriptor construct separates the concept of the physical implementation of a file or a device from the program, one could adopt a communication message descriptor to separate the concept from the actual communication implementation. The message descriptor could be used to implement:

– a piece of information that is shared in memory and cyclically updated,

– a streamed interface.

Below, one finds an example of a possible interface that combines the messaging with a protocol. The set of data type cms_msg_t and corresponding functions cms_open, cms_send, cms_close constitute a portion of a communication management system (cms).

```
moduleA() {
    cms_msg_t msg;
    us_tlc_set_traverse_rate_msg_t rate =
                         { US_TLC_SET_TRAVERSE_SPEED,
                                      0 } ;
    msg.name = "TPS_TO_TLC:TOOL:A"
    msg.protocol = SHMEM;
    cms_open(&msg);
    speed.value = 30; // rpm
    cms_send(msg, rate);
    cms_close(msg);
```

In the example, one opens a communication channel much like one opens a file descriptor within C. In C, the file descriptor can be to a device or a file. Within UTAP Interface Framework, one should assume that the communication descriptor should allow any number of communication protocols, for example, shared memory or INET sockets.

D.3 Configuration File Example

Table D.1 illustrates an example REMOTE configuration file. Within the REMOTE configuration, one can safely assume that single-instance modules are unique, but one may need to establish existence. The enable field defines whether a module exists. For example, a REMOTE system may only consist of an operator joystick interface to the LOCAL system. In this case, most modules in the REMOTE system would be inactive.

The multi-instance modules require a superior/subordinate link to establish the grouping relationship. The multi-instance modules will implicitly be enabled if they are linked to subordinates. The subordinates are grouped according to the UTAP architecture. That is, one cannot expect to group REMOTE modules within a LOCAL subordinate group. Table D.2 illustrates the format required for Figure 3 for the LOCAL system configuration.

D.4 Example of Message Flow for Sample UTAP Scenario

An example of message traffic in a sample scenario will help to verify the interface definitions made about the tasks, sensors, object models and part features. Since UTAP applications

Table D.1 – Example Remote System Configuration File

Module Set	Enabled	Types
OI	Y	Panel
OM, OC, OK	Y	Vanilla
TD, TK, TDS	Y	Teach, Programmable
SGD	Y	see [SANCHO]
ADS	N	
SS	N	
PTPS:TPS:A	Y	Manipulation
PTPS:TPS:B	Y	Navigation
PTPS:TPS:C	Y	Tooling

stress operator-supervised, telerobotic activity, the operator panel is fundamental to assessing strengths and weaknesses. A simple operator panel is given in Figure reffg: oipanel. This panel visually depicts one of many possible operator interface paradigms. This operator paradigm uses the display to do feature-based tooling. The operator chooses from the variety of feature panels (shape, pattern, edge, force) to select the desired parameters. It is assumed that defaults could already be registered on the screen for a particular task. The operator would then select specific feature icons to modify and assist in developing the feature-based world model.

The panels match the object analysis. For example, in the Shape Panel, the panel items have the following meaning - from the top left, clockwise around the panel: target select, 2D circle, 2D rectangle, straight-line, 3D cylindrical volume, 3D cubic volume, obstacle, and a 2D polygon. The pattern panel items correspond to horizontal raster, vertical raster, concentric circular fill, overlap, dither, and orbital. The nozzle panel items correspond to density of spray or flow rate - in one possible data view. If one selects to do an edge instead of pattern fill, the edge panel is available for this task. The edge panel allows exact motion along the curvilinear edge, a cosine weave pattern along the edge (e.g., for welding), and dither correspond to the types of motion along the edges of the part features discussed within the features analysis.

A UTAP sample session is described herein for a refurbishing task. The operator turns on the system. The operator waits for a prompt from the system to proceed. The operator defines a work area by teaching the robot points about the edges of the work area. The operator uses the joystick and moves the robot to each desired location and presses a button to record the location. The work area is usually a default geometry (circle, rectangle). The operator adjusts the parameter settings that are specific to the process. Each process maintains a standoff distance though it varies from process to process. The operator presses a button to start the robot. The robot will move through the taught geometry. The operator observes the process for correct execution. During this time the operator can adjust the parameters as needed. He can pause the process if something is not operating properly (e.g., clogged sprayer). The operator can also press an emergency stop button if something is very wrong. When the process is complete, the operator inspects the results. If there are areas that were not done properly the operator can do a touch-up operation. The operator can do the touch-up himself by moving the robot and controlling the tool or he can define a work area around the bad region and have the robot do it as it did the larger region. Once the work area or part is finished the operator moves

Table D.2 – Example Local System Configuration File

Superior Module Path	Subordinate Module Name	Type
PTPS	TPS:A	Manipulation
	TPS:B	Navigation
	TPS:C	Tooling
TPS:A	TLC:A	Position, Force
		or Compliance, Impedance
TPS:B	TLC:B	Teleop, Guided, Autonomous
TPS:C	TLC:C	Position, Force, Impedance
TLC:A	ROBOT:A	MANIPULATION
	ROBOT:B	
	SENSOR:A	
	TOOL:A	
	SENSOR:B	FTS, IMAGE, PROBE
	TOOL:B	SPRAY
	VS	
TLC:A:SENSOR:B	BEAM BREAK	Switch
TLC:A:TOOL:B:	GRIPPER	grasp
TLC:A:ROBOT:A	RRC	position
TLC:A:ROBOT:B	ACTIVE TOOL	
TLC:A:SENSOR:A	CAMERA	
TLC:A:TOOL:A	3-FINGER GRIPPER	
TLC:A:VS	Proximity	(don't care if sonar or laser)
TLC:B:	ROBOT	TRANSPORT (Lift)
TLC:C	TOOL:A, SENSOR:A	
	TLC:C:TOOL:A	ORBITAL SANDER
TLC:C:SENSOR:A	CAMERA (stationary)	
TLC:C:	ROBOT, SENSOR	
TLC:C:ROBOT	COMPLIANT_ROBOT	
TLC:C:SENSOR	WRIST FORCE SENSOR	

to the next work area or part.

The following message flow summarizes the correspondence between a task steps and message traffic during a refurbishing task. Within the following message flow summary, the channel across which the message is transmitted is listed first. The transmission channel is labelled source_to_destination, where source and destination correspond to the communicating modules. Then, a UTAP message follows optionally requiring calling parameters. Messages that cause recursive action and subsequent messaging before the next step can continue are indented.

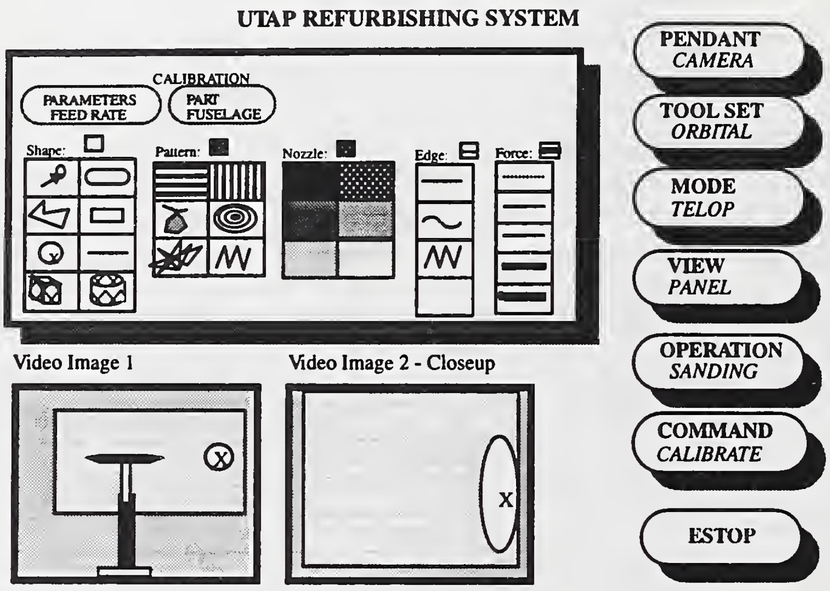

Figure D.2 – Example OI Control Panel

Table D.3 – Sample Session - init

HUMAN_TO_OI	powerup
OI_TO_OC	US_STARTUP(config)
OI_TO_OM	US_STARTUP(config)
OM_TO_OK	US_STARTUP(config)
OI_TO_TD	US_STARTUP(config)
OI_TO_TDS	US_STARTUP(config)
TDS_TO_TK	US_STARTUP(config)
TDS_TO_OK	TPS.A=US_GET_SELECTION_ID("SUBSYSTEM_TPS.A");
TDS_TO_PTPS	US_STARTUP(config)
PTPS_TO_OK	TPS.A=US_GET_SELECTION_ID("SUBSYSTEM_TPS.A");
TDS_TO_TPS.A	US_STARTUP(config)
TDS_TO_OK	TLC.A=US_GET_SELECTION_ID("SUBSYSTEM_TPS.A");
	RSC.A=US_GET_SELECTION_ID("ROBOT A"):
	TC.A=US_GET_SELECTION_ID("TOOL A"):
	TC.A=US_GET_SELECTION_ID("SENSOR A"):
	SC.B=US_GET_SELECTION_ID("SENSOR B"):
TPS.A_TO_TLC.A	US_STARTUP(config)
TLC.A_TO_OK	RSC.A=US_GET_SELECTION_ID("ROBOT A"):
	US_USE_SELECTION(RSC.A);
	US_STARTUP(config)
TLC.A_TO_OK	TC.A=US_GET_SELECTION_ID("TOOL A"):
	US_USE_SELECTION(TC.A); US_STARTUP(config)
TLC.A_TO_OK	SC.A=US_GET_SELECTION_ID("SENSOR A"):
	US_USE_SELECTION(SC.A); US_STARTUP(config)
TLC.A_TO_OK	SC.B=US_GET_SELECTION_ID("SENSOR B"):
	US_USE_SELECTION(SC.B); US_STARTUP(config)
HUMAN_TO_OI	enters Name and Passwd
OI_TO_TDS	US_TDS_LOAD_USER(OPERATOR)
	" System Initialization"
TDS_TO_TK	US_USE_SELECTION_ID, US_TK_GET_FRAMEWORK
TK_TO_TDS	US_POST_FRAMEWORK(...)
TDS_TO_PTPS	US_USE_SELECTION_ID(id for SUBSYSTEM.A) ;
	US_USE_FRAMEWORK("defaults")
PTPS_TO_TPS.A	US_USE_SELECTION_ID(id for SUBSYSTEM.A) ;
	US_USE_FRAMEWORK("defaults") ;
TPS.A_TO_TLC.A	US_USE_SELECTION_ID(id for RSC.A)
	US_BEGIN_MACRO("default setup")
	US_LOAD_DOF(6);
	US_LOAD_REPRESENTATION(Euler);
	US_LOAD_LENGTH_UNITS(mm);
	US_USE_CARTESIAN_MODE (ALL);
	US_USE_ KINEMATIC_RING(_BASE — _TOOL);
	("same as" US_USE_TOOL_TIP_REFERENCE_FRAME();)
	US_LOAD_BASE_PARAMETERS(...);
	US_USE_SELECTION_ID(id for TC.A);
	US_LOAD_TOOL_PARAMETERS(....);
	US_END_SELECTION;
	US_LOAD_JOINT_LIMITS(...);
	US_LOAD_VELOCITY_LIMIT(velmax);
	US_LOAD_ACCELERATION_LIMIT(accmax);
	US_LOAD_TRAVERSE_RATE(tr); US_LOAD_FEED_RATE(fr);
	US_END_MACRO
	US_USE_MACRO("default setup");
TLC.A_TO_RSC.A	US_USE_SELECTION_ID(id for robot servos A);
	US_BEGIN_MACRO("default setup");
	US_LOAD_JOINT_LIMIT(...);
	US_LOAD_VELOCITY_LIMIT(...);
	US_LOAD_PID_GAIN(p,i,d);
	US_USE_ABSOLUTE_POSITION_MODE()
	US_USE_RADIAN_UNITS();
	US_USE_PID();
	"Closed loop control - feedback every 10 milliseconds"
	US_GET_POSITION(ACTUAL, 10ms);
	US_END_MACRO;
	US_END_SELECTION;

Table D.4 – Start Teleoperation

	"Assume lift in place"
HUMAN_TO_OI	"selects subsystem A to do shared control to teach positions"
OI_TO_TDS	US_SELECT_MODE(shared, x-axis)
TDS_TO_TK	US_USE_SELECTION_ID
	US_TK_GET_FRAMEWORK("standoff teach")
TK_TO_TDS	US_POST_FRAMEWORK(...)
TDS_TO_PTPS	US_USE_SELECTION_ID(id for SUBSYSTEM.A);
	US_USE_FRAMEWORK("standoff teach")
PTPS_TO_TPS.A	US_USE_SELECTION_ID(id for SUBSYSTEM.A);
	US_USE_FRAMEWORK("standoff teach") ;
TPS.A_TO_TLC.A	US_USE_SELECTION_ID(id for RSC.A)
	US_BEGIN_MACRO("standoff teach")
	US_USE_MACRO("defaults");
	US_START_STANDOFF_MOTION("x", 300mm);
	US_START_MANUAL_MOTION(ALL X_AXIS);
	US_END_MACRO
	US_USE_MACRO("standoff teach");
HUMAN_TO_OI	push start button, waits for robot to home, then use joystick
OI_TO_TDS	US_START
TDS_TO_TPS.A	US_START
TPS.A_TO_TLC.A	US_HOME; US_START;
TLC.A_TO_RSC.A	US_USE_MACRO("default");
	US_HOME; // put values in, for eventual motion
	US_ENABLE(ALL); // enable servos
	US_CLEAR_BRAKES(ALL);
	US_START; // software start
TLC.A_TO_SEN.A	US_START_MACRO("range")
	US_LOAD_SAMPLING_SPEED(speed)
	US_LOAD_FREQUENCY(freq)
	"more sensor inits?"
	US_END_MACRO
	US_USE_MACRO("range")
	"Repeated Joystick Motion"
HUMAN_TO_OI	6 DOF joystick motion
OI_TO_TLC.A	US_BEGIN_BLOCK; US_ADJUST_AXIS(ALL, values);
	US_END_BLOCK;
	"Monitors for standoff distance"
OI_TO_TLC.A	US_SET_POSITION(desired_values)
TLC.A_TO_RSC.A	US_SET_POSITION(desires_values);
RSC.A_TO_TLC.A	US_POST_POSITION(actual_values)
	"Calibrate Corner of Rect"
HUMAN_TO_OI	marks corner of feature (where feature = shape + pattern)
OI_TO_OC	US_SET_CALIB(feature_origin, rect);
OC_TO_OK	now=US_GET_POSITION(actual);
	rect.x=now.y; rect.y= now.z;
OC_TO_OK	obj=US_CREATE_OBJECT(name, part_t, rect, raster);
	"Move to Second Point using Joystick Motion - see above"
	"Define Second Corner"
HUMAN_TO_OI	marks 2nd corner of feature (where feature = shape + pattern)
OI_TO_OC	US_SET_CALIB(feature_offset, rect);
OC_TO_OK	now=US_GET_POSITION(actual);
	obj.xlength = rect.x - now.y;
	obj.ylength = rect.y - now.z;
OC_TO_OK	US_MODIFY_OBJECT(obj);
	"User finished teaching"
HUMAN_TO_OI	presses button to end teaching
OI_TO_TDS	US_STOP
TDS_TO_TPS	US_STOP
TPS_TO_TLC	US_BEGIN_MACRO("standoff teach halt")
	US_STOP_MANUAL_MOTION
	US_STOP_STANDOFF_MOTION
	US_END_MACRO
	US_USE_MACRO("standoff teach halt")

Table D.5 – Start Automated Process

	"Start Automated Process"
HUMAN_TO_OI	"human presses button to get into process control screen"
OI_TO_TDS	US_SELECT_MODE(supervised, all)
OI_TO_TDS	US_SET_TDS_SELECT_OPERATION("strip")
TDS_TO_TK	US_USE_SELECTION_ID, US_TK_GET_FRAMEWORK("strip")
TK_TO_TDS	US_POST_FRAMEWORK(...)
TDS_TO_PTPS	US_USE_SELECTION_ID(id for SUBSYSTEM.A) ;
	US_USE_FRAMEWORK("standoff teach")
PTPS_TO_TPS.A	US_USE_SELECTION_ID(id for SUBSYSTEM.A) ;
	US_USE_FRAMEWORK("standoff teach") ;
TPS.A_TO_TLC.A	US_USE_SELECTION_ID(id for RSC.A)
	US_BEGIN_MACRO("standoff teach")
	US_USE_MACRO("defaults");
	US_START_STANDOFF_MOTION("x", 300mm);
	US_START_MANUAL_MOTION(ALL X_AXIS);
	US_END_MACRO
	US_USE_MACRO("standoff teach");
TLC.A_TO_SEN.A	US_START_MACRO("range")
	US_LOAD_SAMPLING_SPEED(speed)
	US_LOAD_FREQUENCY(freq)
	US_END_MACRO
	US_USE_MACRO("range")
TLC.A_TO_TOOL.A	US_FLOW_LOAD_PARAMETERS(flow_rate, beam, stream)
HUMAN_TO_OI	"human presses button on OI to start process"
OI_TO_TDS	US_START();
TDS_TO_PTPS	US_USE_SELECTION_ID(id for SUBSYSTEM.A) ; US_START();
PTPS_TO_TPS.A	US_USE_SELECTION_ID(id for SUBSYSTEM.A) ; US_START();
TPS.A_TO_TLC.A	US_USE_SELECTION_ID(id for RSC.A)
	US_START();
TLC.A_TO_SEN.A	US_ENABLE()
	US_START();
TLC.A_TO_TOOL.A	US_ENABLE();
	US_START();
TLC.A_TO_ROBOT.A	US_ENABLE();
	US_CLEAR_BRAKES();
	US_START();
	"Subsystem TLC.A Series of commands to do raster path"
SEN.A_TO_TLC.A	US_POST_READING("range reading")
TLC.A_TO_ROBOT.A	US_AXIS_SERVO_SET_POSITION("desired position");
ROBOT.A_TO_TLC.A	US_POST_READING("actual position")
	US_GENERIC_STATUS_REPORT(executing, progressing)
	...
	US_GENERIC_STATUS_REPORT(finished, succeeded);

Annex E
(informative)
Related Standards

E.1 RS274D

EIA/RS274D is a standard programming language that is intended to serve as a uniform interface for command and control of numerically controlled machine tools.

E.2 RS441

The UTAP operator interfaces will use the RS441 existing standard to define operator control and modes of operation.

E.3 POSIX

ISO/IEC 9945 and IEEE 1003 standard series are intended to define a standard portable operating system interface and environment to support application portability at the source-code level. Areas of POSIX standardization efforts include definitions for system services; user interface (shell) and associated commands; real-time extensions; networking protocols; graphical interfaces; data base management system interfaces; object and binary code portability; system configuration and resource availability; behavior of system services for systems supporting concurrency within a single process.

E.4 IEC 1131-3

Parent Task Program Sequencing input *shall* use IEC1131 Part 3 as a the interface language to describe any parallel or simultaneous behavior.

IEC 1131 Part 3 specifies the syntax and semantics of a unified suite of programming languages for Programmable Controllers (PCs). These consist of two textual languages, IL (instruction lists) and ST (Structured Text) and two graphical languages LD (Ladder Diagram) and FBD (Function Block Diagram). Sequential Function Chart (SFC) elements are defined for structuring the internal organization of PC programs and function blocks written in one of the 4 languages.

The SFC elements provide a means of partitioning a PC program organization unit into a set of steps and transitions inter-connected by directed links. Associated with each step is a set of actions, and with each transition is associated a transition condition. Because SFC elements require storage of state information, the only program organization units which can be structured using these elements re function blocks and programs. Configuration elements are defined which support the installation of PC programs into PC systems and include configurations, resources, tasks, global variables, and access paths. A configuration contains one or more resources (e.g., CPU) each of which may contain one or more tasks and programs

E.5 ANSI/RIA R15-06-1992

The American National Standard (ANSI) for Industrial Robots and Robot Systems Safety Requirements, ANSI/RIA R15-06-1992, Sponsor: RIA was approved - August 19, 1992. The purpose of this standard is to provide guidelines for industrial robot manufacture, remanufacture and rebuild; robot system installation; and methods of safeguarding to enhance the safety of personnel associated with the use of robots and robot systems."

E.6 EIA Standard RS-267-A

This standard comply with terminology defined in EIA/RS267-A for "Axis and Motion Nomenclature for Numerically Controlled Machines."

E.7 XDR

Public-domain set of routines to allow C programmers to describe arbitrary data structures in a machine-independent fashion. Data for remote procedure calls (RPC) are encoded and decoded using XDR. Can be used for other heterogeneous platform communication as well.

Annex F
(informative)
Target Applications

The architecture has been developed for general aircraft maintenance and remanufacturing applications. Among the many applications in aircraft maintenance and remanufacturing, three target applications were specifically addressed: stripping paint from the skin of an aircraft; surface finishing; and advanced cutting. The potential application of telerobotics to these applications is described in this section.

F.1 Paint Stripping

One way of stripping paint from the skin of an aircraft is to blast Plastic Media Bead (PMB) on the painted surface of the aircraft. The operator applies PMB to a targeted surface area with a certain pressure, using the blast gun located at a designated distance from the surface with a certain orientation (relative to the tangential plane of the surface). To cover the entire surface area of an aircraft, a mobile platform or a telecrane is used to move the operator around the aircraft. More specifically, the paint stripping task requires the following subtasks and considerations:

a) The positioning of the mobile platform at a location that allows the operator to cover the new targeted area with sufficient dexterity.

b) The maintaining of the designated distance and orientation of the gun with respect to the blasting surface, while following the proper trajectory.

c) The controlling of the speed of the gun based on the visual monitoring of the progress of stripping. Due to the possible difference in paint thickness, without proper monitoring of the progress of stripping for adjusting speed and pressure of blast, over-stripping as well as under-stripping may result. The skill of the operator is important for this task.

d) The finishing up process to strip under-stripped areas.

It is expected that the application of telerobotics to the above paint stripping task can bring forth the following advantages:

a) The operator can stay in a remote location protected from pollutant contamination during operation, such that not only safety but also efficiency in task execution can be enhanced.

b) The machine may be better in accuracy and consistency for maintaining the distance and orientation of the gun with respect to the blasting surface.

c) The larger workspace of manipulators can be exploited.

d) The advanced visual sensors and displays may provide the operator with more effective tools for inspecting the progress of stripping.

Based on the above observations, we can construct the following telerobotics system for the paint-stripping operation:

a) A dextrous manipulator replaces a human operator in the immediate worksite.

b) The human operator is able to manually control the manipulator.

c) The human operator is given visual displays for monitoring the progress of paint stripping. The visual displays may be based on cameras mounted on the manipulator or based on another manipulator carrying cameras and light sources and other sensors.

d) Sensor-based automatic operations are provided for maintaining the distance and orientation of the gun automatically.

e) The manipulator trajectories can be determined by the human operator, or by the system, or by a combination of both. The trajectories generated by the system can be from the off-line interactive graphic simulation or from a functional form in relation to the known geometric model of the target surfaces. The capability of combining the manual and system trajectories allows the integration of the operator skill in reacting to the visual monitoring of the task progress.

f) To execute the system generated trajectories, the manipulator should be registered on a predetermined location or localized with respect to the geometric model of the surface.

g) The application program developer should be provided with an iconic and menu-driven interface that allows easy programming. That is, programming is done by configuring the existing software modules through an iconic and menu-driven interactive computer interface.

F.2 Telerobotic Surface Finishing

Surface finishing is an important task in aircraft maintenance and remanufacturing. The damaged or corroded portion of the aircraft skin is patched or replaced. Uneven surfaces are ground smooth. Telerobotics technology can be used for automatically controlling the contact force/torque of a tool during surface finishing while maintaining the designated tool angles with respect to the surface normal, without à priori knowledge of part geometry, through shared control. The tool path may be generated either manually by the operator or from the preassigned trajectory generated by off-line programming. The tool path may be subject to certain artificial motion constraints. Note that, in the case of manual operation, the contact force/torque needs to be guarded so as not to exceed the maximum allowable force/torque. Similar to the paint stripping task, the operator should monitor the progress of surface finishing based on visual and graphic displays, so that the operator can fine-tune or modify the tool path accordingly.

The surface finishing task seems quite different from the paint stripping task. However, a common telerobotics architecture can be used. This is because both tasks are based upon the shared and cooperative control between human and machine, in spite of the fact that the surface finishing task depends on force/torque sensing whereas the paint stripping task depends on proximity sensing for sensor-based automatic operations. The only major difference is that the path fusion in surface finishing requires consideration of the increment of force and torque together with the increment of path.

F.3 Telerobotic Advanced Cutting System

The maintenance and remanufacturing of aircraft requires cutting of all types of material in many different shapes and sizes. Telerobotics technology can provide shared control for generating the tool trajectory in advanced cutting. The trajectory may be specified by prestored data generated by off-line programming or by the operator through a hand controller or by visual servoing of a marked path on the object surface. The system automatically regulates the surface stand-off/separation and the tool orientation at the designated values, as well as imposing certain artificial constraints on the trajectory. The change of tools and the initialization of system need to be incorporated into the system.

The telerobotics architecture for advanced cutting is basically same as that of paint stripping and surface finishing. A unique feature is the integration of visual servoing based on vision sensors.

Annex G
(informative)
API Issues

Defining the range of capability expected of the API mechanism is problematic. One cannot arrive at the perfect solution that is exceedingly complex or prohibitively expensive. Instead, compromises must be made in arriving at an API mechanism that resolve issues for flexibility and extensibility. Issues that the UTAP API will have to resolve (noted by the (Unresolved) after the item) or have been resolved (noted by the (Resolved) after the item) include

G.1 Messages, Macros and Naming

The UTAP interfaces will define a broad API. Yet, it would be impossible to anticipate and explicitly enumerate every possible control and sensing algorithm and parameter. For example, suppose a better control algorithm is developed, how will the interface permit the selection of this algorithm? Further, suppose an additional compensation parameter could be specified within the servo control. How will the system adapt to the additional parametric capability? Will macro programmability of an interface be allowed, and how would this be achieved?

G.2 Integration

One desires the ability to do on-line configuration and assignment of modules and connect the module communication. The ability to CONNECT/DISCONNECT to actual devices (such as actuators or sensors) or virtual devices (such as other modules) is provided by the UTAP API definitions. Once connected, one must be concerned with communication data flow.

The connection for command communication (such as a superior-subordinate connection) is straightforward. In this case, one sends goal-action commands to a subordinate and awaits results. However, model-driven data communication (peer-to-peer) is not directly apparent. For example, when you add a new sensor to your system, how do you pump this sensor data into an Trajectory Generation module for dynamic path modification. Receiving data from a connection is straightforward. One connects to the module, queries a variable and reads the updates that the module provides. Unresolved is the application of external data within a module. UTAP API provide externally-accessible model-driven variables within UTAP API modules for update - such as overrides or offsets - to allow integration of user-customized or third-party sensor-based control. (The question remains whether enough externally-accessible model-driven variables will be defined.)

G.3 Definition Style

The style of the API definitions is of considerable importance. One could use ASN.1 or the STEP modeling language EXPRESS to develop a rigorous definition of the interfaces. For all indications from UTAP API members, this is be too cumbersome and approach. One could use BNF, source headers files, or any syntax definition mechanism to define a grammar that each channel accepts. There is a trade-off between interface language complexity and performance. This is an issue of major importance that has been discussed but every solution has baggage.

This Definition Style issue also must resolve the problem of differentiating between cyclically executing processes (such as servo or discrete logic module or trajectory generator) and asynchronous processes (such as Part Program Interpreter and the Task Coordination Module.) The problem is that a definition consisting of a set of function calls alone is not sufficient to describe the a cyclically executing or event-driven API. One needs to understand the trigger mechanism that drives the event (such as an external clock or a synchronization trigger from a cooperating module.) This issue has been discussed but no final resolution has been forwarded.

G.4 Variable Length Arrays

One of the problems that arises defining interfaces concerns the handling of variable length arrays. Unless one rejects the notion of flexibility, an interface should not preordain a fixed array size for any interface. One would find passing 5 axis values to a 3-axis mill less than intuitive. Heaps will be used to resolve this problem.

G.5 Units and Representation

It is possible to mandate Standard International Units. Yet, this can cause problems since one prefers to use units that are natural for the application (millimeters, inches, etc.) For many robotics and automation applications, the millimeter is the natural and intuitive way of thinking about a problem. There should be no reason to contradict the natural reasoning process. Further, NASA mandates foot and pounds as the units of choice. Thus, one needs conversion. One has to make the decision as to whether the conversion is done by the sender or the receiver. In the vendor marketplace, a commercial product module should provide conversion utilities.

UTAP modules *shall* state acceptable measurement units within its interfaces. The range of acceptable measurements units *may* become broader as the application requirements dictate. For example, an automated horse may require the addition of the furlong distance as an interface measurement unit.

The default units *shall* be SI, and are:

From a standards aspect, data exchange between modules is designed to be in a neutral representation. However, selection of the correct neutral representation is also problematic. The UTAP modules *shall* support API definitions with selectable representation as a part of the mode control.

NOTE 1 – The UTAP interfaces include the representation measurement units of an interface item. Incompatibility among like-representation, dissimilar-units interfaces will be resolved by providing use_measurement_units or use_representation_type messaging. If the module does not support the measurement units or representation types that you desire, the programmer must perform the conversion. It is assumed that more robust modules will be better able to handle a broader variety of representation units and be ultimately more commercially viable. For example, a trajectory interface may accept trajectory position descriptions in millimeters or meters or even inch length units. Or the trajectory may accept orientation represented as Euler angles in degrees or radians or as elements in a Homogeneous Matrix representation.

Table G.1 – Parameter and Units

Distance or Length or Position–	Meters
Velocity–	Meters/Second
Acceleration–	Meters/Second^2
Jerk–	Meters/Second^3
Angular measurement–	Radians
Forces–	Newtons
Torques–	NewtonMeter
Light–	Lumen
Viscosity–	millipascalSecond (mPa S)
Humidity–	Grams/Meter^3 "That's grams of water"
Temperature–	Celsius
Noise–	Decibel

G.6 Selection

Multiple subordinate modules to be controlled by one superior module is possible. Because of the existence of alternatives, some messages to a subordinate can be ambiguous as to their intent. In the case of multiple axes of control, one must resolve the destination for which axis the command is intended. The framework will provide a device/module naming convention but the selection mechanism is unresolved.

G.7 Parameterization

At opposite ends of the spectrum is a programming facility with a large set of functions and fixed parameters versus a programming facility with small set of functions and a wider range of arguments. The information content is the same. Yet, the presentation and programming is different. As for comprehension, there are arguments for and against both styles. For example, source is given below for the range of styles.

```
#define NML_SERVO_SET_ABS_POSITION 251
struct  nml_servo_set_abs_position_msg_t {
    int msgid;
    double *joint_position;
};

#define NML_SERVO_SET_REL_POSITION 252
struct  nml_servo_set_rel_position_msg_t {
int msgid;
double *joint_position;
};

#define NML_SERVO_JOG 257
struct  nml_servo_jog_msg_t {
    int msgid;
    int axis;
    double speed;
};

#define NML_SERVO_ADJUST_AXIS 655
struct us_tlc_adjust_axis_msg_t {
    int msgid;
```

```
        int axis;          // axis mask
        double *increment; // if amount=0, system decides
    };

    Style 2: Embed modes:
    ===========================================
    #define NML_SERVO_SET_POSITION 251
        struct  nml_servo_set_position_msg_t {
            int msgid;
            enum {absolute,
                  relative,
                  incremental,
                  jog,   // may not belong
               } mode;
            double *update;
    };
```

To further cloud the issue one can turn both the mode and the parameter into arguments. For example, one can set both the mode and parameter type be it position, velocity or acceleration.

```
    Style 3: Arguments
    ===========================================
    #define NML_SERVO_SET_VALUE 251
    struct nml_servo_set_msg_t {
        int msgid;
        enum {absolute,
              relative,
              incremental,
              jog,   // may not belong
        } mode;
        enum {position,
              velocity,
              acceleration,
        } parameter;
        double *update;
};
```

The last case is more concise, however, unless all combinatorial arguments states are valid, illegal and illogical messages can be formed. For example, does jogging the acceleration make sense? The UTAP interfaces are currently defined with a larger set of functions to allow scaling within this mechanism, although discussions are ongoing as to the efficacy of this method.

G.8 Aggregation Model

One of the issues effecting the specification of open architectures is the approach to connecting modules.

– Consistent approach wherein explicit module exists to translate from one level of functionality to another level of functionality. This module may have zero (or phantom/hidden) functionality, in that, its only capability is to translate from a representation at a higher level of abstraction into representation at a lower level of abstraction.

The major benefit to this approach is a consistent paradigm that simplifies interfaces between modules to a more manageable set and offers explicit scalability and interoperability, in that, a direct swap of modules without "rewiring" can be used to improve performance.

The major drawback to this approach is that it is at first counterintuitive, and second may appear to add too much overhead in the communication.

– Free-wiring allows interface traffic from a high level of abstraction to any low level of abstraction. Thus, not all modules are necessary when building a system. However, this method assumes that a higher level module understands the needs and representation of a lower level module. The drawback is that upgrading the controlling by adding modules to improve capability is not straightforward.

Annex H
(normative)
Interface Descriptions

The interfaces were defined as a set of messages. Each message has an unique numeric identifier and data structure defining parameter values. This annex contains the list of interface messages sorted by module by type and alphabetically, as well as the current interface definitions.

The C/C++ language was used to define messages. This annex gives source listings of the header files used to define the interfaces. The header files are given in the following order:

- *utap_disclaimer.h*
- *generic_defs.h*
- *utap_classification.h*
- *utap_info_model.h*
- *utap_protocol.h*
- *utap_data_defs.h*
- *utap_interfaces.h*
- *utap_api.h*

The interfaces defined with API function calls were generated by a shell script that translated the messages data structures into function prototypes. The **enum** and **union** C++ constructs did not have direct mapping within the calling function, so placeholders were used.

The information models, messages, and function prototypes in the header files were compiled with a GNU gcc version 2.5.8 a variant of C++. The code may look like C, but it is actually C++.

H.1 Interface List

```
UTAP_INTERFACE_DEFINITIONS
GENERIC
    US_STARTUP
    US_SHUTDOWN
    US_RESET
    US_ENABLE
    US_DISABLE
    US_ESTOP
    US_START
    US_STOP
    US_ABORT
    US_HALT
    US_INIT
    US_HOLD
    US_PAUSE
    US_RESUME
    US_ZERO
    US_BEGIN_SINGLE_STEP
    US_NEXT_SINGLE_STEP
    US_CLEAR_SINGLE_STEP
    US_BEGIN_BLOCK
    US_END_BLOCK
    US_BEGIN_PLAN
    US_END_PLAN
    US_USE_PLAN
    US_BEGIN_MACRO
    US_END_MACRO
    US_USE_MACRO
    US_BEGIN_EVENT
    US_END_EVENT
    US_MARK_BREAKPOINT
    US_MARK_EVENT
    US_GET_SELECTION_ID
    US_POST_SELECTION_ID
    US_USE_SELECTION
    US_USE_AXIS_MASK
    US_USE_EXT_ALGORITHM
    US_LOAD_EXT_PARAMETER
    US_GET_EXT_DATA_VALUE
    US_POST_EXT_DATA_VALUE
    US_SET_EXT_DATA_VALUE
    US_LOAD_STATUS_TYPE
    US_LOAD_STATUS_PERIOD
    US_GENERIC_STATUS_REPORT
ERRORS
    US_ERROR_COMMAND_NOT_IMPLEMENTED
    US_ERROR_COMMAND_ENTRY
    US_ERROR_DUPLICATE_NAME
    US_ERROR_BAD_DATA
    US_ERROR_NO_DATA_AVAILABLE
    US_ERROR_SAFETY_VIOLATION
    US_ERROR_LIMIT_EXCEEDED
    US_ERROR_OVER_SPECIFIED
    US_ERROR_UNDER_SPECIFIED
AXIS_SERVO
    US_AXIS_SERVO_USE_ANGLE_UNITS
    US_AXIS_SERVO_USE_RADIAN_UNITS
    US_AXIS_SERVO_USE_ABS_POSITION_MODE
    US_AXIS_SERVO_USE_REL_POSITION_MODE
    US_AXIS_SERVO_USE_ABS_VELOCITY_MODE
    US_AXIS_SERVO_USE_REL_VELOCITY_MODE
    US_AXIS_SERVO_USE_PID
    US_AXIS_SERVO_USE_FEEDFORWARD_TORQUE
    US_AXIS_SERVO_USE_CURRENT
    US_AXIS_SERVO_USE_VOLTAGE
    US_AXIS_SERVO_USE_STIFFNESS
    US_AXIS_SERVO_USE_COMPLIANCE
    US_AXIS_SERVO_USE_IMPEDANCE
    US_AXIS_SERVO_START_GRAVITY_COMPENSATION
    US_AXIS_SERVO_STOP_GRAVITY_COMPENSATION
    US_AXIS_SERVO_LOAD_DOF
    US_AXIS_SERVO_LOAD_CYCLE_TIME
    US_AXIS_SERVO_LOAD_PID_GAIN
    US_AXIS_SERVO_LOAD_JOINT_LIMIT
    US_AXIS_SERVO_LOAD_VELOCITY_LIMIT
    US_AXIS_SERVO_LOAD_GAIN_LIMIT
    US_AXIS_SERVO_LOAD_DAMPING_VALUES
    US_AXIS_SERVO_HOME
    US_AXIS_SERVO_SET_BRAKES
    US_AXIS_SERVO_CLEAR_BRAKES
    US_AXIS_SERVO_SET_TORQUE
    US_AXIS_SERVO_SET_CURRENT
    US_AXIS_SERVO_SET_VOLTAGE
    US_AXIS_SERVO_SET_POSITION
    US_AXIS_SERVO_SET_VELOCITY
    US_AXIS_SERVO_SET_ACCELERATION
    US_AXIS_SERVO_SET_FORCES
    US_AXIS_SERVO_JOG
    US_AXIS_SERVO_JOG_STOP
TOOL
    US_SPINDLE_RETRACT_TRAVERSE
    US_SPINDLE_LOAD_SPEED
    US_SPINDLE_START_TURNING
    US_SPINDLE_STOP_TURNING
    US_SPINDLE_RETRACT
    US_SPINDLE_ORIENT
    US_SPINDLE_LOCK_Z
    US_SPINDLE_USE_FORCE
    US_SPINDLE_USE_NO_FORCE
    US_FLOW_START_MIST
    US_FLOW_STOP_MIST
    US_FLOW_START_FLOOD
    US_FLOW_STOP_FLOOD
    US_FLOW_LOAD_PARAMETERS
SENSOR
    US_START_TRANSFORM
    US_STOP_TRANSFORM
    US_START_FILTER
    US_STOP_FILTER
    US_SENSOR_USE_MEASUREMENT_UNITS
    US_SENSOR_LOAD_SAMPLING_SPEED
    US_SENSOR_LOAD_FREQUENCY
    US_SENSOR_LOAD_TRANSFORM
    US_SENSOR_LOAD_FILTER
    US_SENSOR_GET_READING
    US_SENSOR_GET_ATTRIBUTES_READING
    US_VECTOR_SENSOR_GET_READING
    US_FT_SENSOR_POST_READING
    US_SCALAR_SENSOR_POST_READING
    US_VECTOR_SENSOR_POST_READING
    US_2D_SENSOR_LOAD_ARRAY_PATTERN
    US_2D_SENSOR_USE_ARRAY_TYPE
    US_2D_SENSOR_GET_READING
    US_2D_SENSOR_POST_READING
    US_IMAGE_USE_FRAME_GRAB_MODE
    US_IMAGE_USE_HISTOGRAM_MODE
    US_IMAGE_USE_CENTROID_MODE
    US_IMAGE_USE_GRAY_LEVEL_MODE
    US_IMAGE_USE_TRESHOLD_MODE
    US_IMAGE_COMPUTE_SPATIAL_DERIVATIVES_MODE
    US_IMAGE_COMPUTE_TEMPORAL_DERIVATIVES_MODE
    US_IMAGE_USE_SEGMENTATATION_MODE
    US_IMAGE_USE_RECOGNITION_MODE
    US_IMAGE_COMPUTE_RANGE_MODE
    US_IMAGE_COMPUTE_FLOW_MODE
    US_IMAGE_LOAD_CALIBRATION
    US_IMAGE_SET_POSITION
    US_IMAGE_ADJUST_POSITION
    US_IMAGE_ADJUST_FOCUS
    US_IMAGE_POST_SPECIFICATION
    US_IMAGE_POST_PIXEL_MAP_READING
    US_IMAGE_POST_HISTOGRAM_READING
    US_IMAGE_POST_XY_CHAR_READING
    US_IMAGE_POST_BYTE_SYMBOLIC_READING
    US_IMAGE_POST_TRESHOLD_READING
    US_IMAGE_POST_SPATIAL_DERIVATIVE_READING
    US_IMAGE_POST_TEMPORAL_DERIVATIVE_READING
    US_IMAGE_POST_RECOGNITION_READING
    US_IMAGE_POST_RANGE_READING
    US_IMAGE_POST_FLOW_READING
PROGRAMMABLE_IO
    US_PIO_ENABLE
    US_PIO_DISABLE
    US_PIO_SET_MODE
    US_PIO_CONTROL_WRITE
    US_PIO_LOAD_SCALE
    US_PIO_DATA_WRITE
    US_PIO_DATA_READ
    US_PIO_BIT_READ
    US_PIO_BIT_SET
    US_PIO_TOGGLE_BIT
    US_PIO_POST_DATA
TASK_LEVEL_CONTROL
    US_TLC_USE_JOINT_REFERENCE_FRAME
    US_TLC_USE_CARTESIAN_REFERENCE_FRAME
    US_TLC_USE_REPRESENTATION_UNITS
    US_TLC_USE_ABSOLUTE_POSITIONING_MODE
    US_TLC_USE_RELATIVE_POSITIONING_MODE
    US_TLC_USE_WRIST_COORDINATE_FRAME
    US_TLC_USE_TOOL_TIP_COORDINATE_FRAME
    US_TLC_CHANGE_TOOL
    US_TLC_USE_MODIFIED_TOOL_LENGTH_OFFSETS
    US_TLC_USE_NORMAL_TOOL_LENGTH_OFFSETS
    US_TLC_USE_NO_TOOL_LENGTH_OFFSETS
    US_TLC_USE_KINEMATIC_RING_POSITIONING_MODE
    US_TLC_START_MANUAL_MOTION
    US_TLC_STOP_MANUAL_MOTION
    US_TLC_START_AUTOMATIC_MOTION
    US_TLC_STOP_AUTOMATIC_MOTION
    US_TLC_START_TRAVERSE_MOTION
    US_TLC_STOP_TRAVERSE_MOTION
    US_TLC_START_GUARDED_MOTION
    US_TLC_STOP_GUARDED_MOTION
    US_TLC_START_COMPLIANT_MOTION
    US_TLC_STOP_COMPLIANT_MOTION
    US_TLC_START_FINE_MOTION
    US_TLC_STOP_FINE_MOTION
    US_TLC_START_MOVE_UNTIL_MOTION
    US_TLC_STOP_MOVE_UNTIL_MOTION
    US_TLC_START_STANDOFF_DISTANCE
    US_TLC_STOP_STANDOFF_DISTANCE
    US_TLC_START_FORCE_POSITIONING_MODE
    US_TLC_STOP_FORCE_POSITIONING_MODE
    US_TLC_LOAD_DOF
    US_TLC_LOAD_CYCLE_TIME
    US_TLC_LOAD_REPRESENTATION_UNITS
    US_TLC_LOAD_LENGTH_UNITS
    US_TLC_LOAD_RELATIVE_POSITIONING
    US_TLC_ZERO_RELATIVE_POSITIONING
    US_TLC_ZERO_PROGRAM_ORIGIN
    US_TLC_LOAD_KINEMATIC_RING_POSITIONING_MODE
    US_TLC_LOAD_BASE_PARAMETERS
    US_TLC_LOAD_TOOL_PARAMETERS
    US_TLC_LOAD_OBJECT
    US_TLC_LOAD_OBJECT_BASE
    US_TLC_LOAD_OBJECT_OFFSET
    US_TLC_LOAD_DELTA
    US_TLC_LOAD_OBSTACLE_VOLUME
    US_TLC_LOAD_NEIGHBORHOOD
    US_TLC_LOAD_FEED_RATE
    US_TLC_LOAD_TRAVERSE_RATE
    US_TLC_LOAD_ACCELERATION
    US_TLC_LOAD_JERK
    US_TLC_LOAD_PROXIMITY
    US_TLC_LOAD_CONTACT_FORCES
    US_TLC_LOAD_JOINT_LIMIT
    US_TLC_LOAD_CONTACT_FORCE_LIMIT
    US_TLC_LOAD_CONTACT_TORQUE_LIMIT
    US_TLC_LOAD_SENSOR_FUSION_POS_LIMIT
    US_TLC_LOAD_SENSOR_FUSION_ORIENT_LIMIT
    US_TLC_LOAD_SEGMENT_TIME
    US_TLC_LOAD_TERMINATION_CONDITION
    US_TLC_INCR_VELOCITY
    US_TLC_INCR_ACCELERATION
    US_TLC_SET_GOAL_POSITION
    US_TLC_GOAL_SEGMENT
    US_TLC_ADJUST_AXIS
    US_TLC_UPDATE_SENSOR_FUSION
    US_TLC_SELECT_PLANE
    US_TLC_USE_CUTTER_RADIUS_COMPENSATION
    US_TLC_START_CUTTER_RADIUS_COMPENSATION
    US_TLC_STOP_CUTTER_RADIUS_COMPENSATION
    US_TLC_STRAIGHT_TRAVERSE
    US_TLC_ARC_FEED
    US_TLC_STRAIGHT_FEED
    US_TLC_PARAMETRIC_2D_CURVE_FEED
    US_TLC_PARAMETRIC_3D_CURVE_FEED
    US_TLC_NURBS_KNOT_VECTOR
    US_TLC_NURBS_CONTROL_POINT
    US_TLC_NURBS_FEED
    US_TLC_TELEOP_FORCE_REFLECTION_UPDATE
TASK_DESCRIPTION
    US_TDS_LOAD_USER
    US_TDS_SELECT_PROGRAM
    US_TDS_EXECUTE_PROGRAM
    US_TDS_SELECT_OPERATION
    US_TDS_SELECT_OPMODE
    US_TDS_LOAD_SELECTIONS
    US_TDS_LOAD_REFERENCE_UNITS
    US_TDS_LOAD_RATE_DEFAULTS
    US_TDS_LOAD_ORIGIN
    US_TDS_LOAD_SENSING_DEFAULTS
TASK_KNOWLEDGE
    US_TK_DEFINE_FRAMEWORK
    US_TK_MACRO_CREATE
    US_TK_MACRO_DELETE
    US_TK_MACRO_MODIFY
PARENT_TASK_PROGRAM_SEQUENCING
    US_PTPS_SELECT_AGENT
    US_TPS_SELECT_TOOL
    US_PTPS_SELECT_SENSOR
    US_PTPS_INTERP_RUN_PLAN
    US_PTPS_INTERP_HALT_PLAN
    US_PTPS_INPUT_REQUEST
    US_PTPS_OUTPUT_ENABLE_SUBSYSTEM
TASK_PROGRAM_SEQUENCING
    US_TPS_FREESPACE_MOTION
    US_TPS_GUARDED_MOTION
    US_TPS_CONTACT_MOTION
    US_TPS_SET_SUPERVISORY_MODE
    US_TPS_SELECT_FEATURE
    US_TPS_SELECT_MATERIAL
    US_LOAD_OBSTACLE
    US_LOAD_PATTERN
    US_TPS_MARK_EVENT
    US_TPS_ENABLE
OPERATOR_INTERFACE
    US_BEGIN_FRAMEWORK
    US_END_FRAMEWORK
    US_CREATE_FRAMEWORK
    US_DELETE_FRAMEWORK
    US_ADD_SYMBOLIC_ITEM
    US_DELETE_SYMBOLIC_ITEM
    US_ADD_SYMBOLIC_ITEM_ATTR
    US_DELETE_SYMBOLIC_ITEM_ATTR
    US_SET_SYMBOLIC_ITEM_ATTR
OBJECT_MODELING
    US_OM_CREATE
    US_OM_DELETE
    US_OM_MODIFY
OBJECT_CALIBRATION
    US_OC_SET_CALIB
    US_OC_GET_CALIB
    US_OC_SET_ATTR
    US_OC_GET_ATTR
OBJECT_KNOWLEDGE
    US_OK_RECORD
    US_OK_PLAYBACK
    US_OK_CREATE_OBJ
    US_OK_DELETE_OBJ
    US_OK_MODIFY
    US_OK_MODIFY_ATTRIBUTE
    US_OK_ATTRIBUTE_QUERY
    US_OK_OUTPUT_REGISTERED_OBJ_ID
    US_OK_ATTRIBUTE_RESPONSE
TRAJECTORY_DESCRIPTION
    US_TRD_OPEN
    US_TRD_ERASE
    US_TRD_RECORD
    US_TRD_RECORD_ON
    US_TRD_RECORD_OFF
    US_TRD_FIND
    US_TRD_NEXT
    US_TRD_PREVIOUS
    US_TRD_DELETE
    US_TRD_NAME_ITEM
    US_TRD_DELETE_ITEM
    US_TRD_SET_JOINT_MODE
    US_TRD_SET_CARTESIAN_MODE
    US_TRD_MODIFY
    US_TRD_ADD_ELEMENT
STATUS_GRAPHICS_DISPLAY
ANALYSIS_DIAGNOSIS_SYSTEM
    US_ADS_COLLISION_DETECTED
SUBSYSTEM_SIMULATION
UTAP_DATA_DEFS
    US_POST_ID
    US_GET_OBJECT_ID
    US_USE_OBJECT
    US_GET_FEATURE
    US_USE_FEATURE
    US_GET_VALUE
    US_POST_VALUE
    US_GET_LIST
    US_POST_LIST
    US_ATTRIBUTE_POST_RESPONSE
    US_ATTRIBUTE_GET_TIME
    US_ATTRIBUTE_GET_POSITION
    US_ATTRIBUTE_GET_ORIENTATION
    US_ATTRIBUTE_GET_POSE
    US_ATTRIBUTE_GET_VELOCITY
    US_ATTRIBUTE_GET_ACCELERATION
    US_ATTRIBUTE_GET_JERK
    US_ATTRIBUTE_GET_FORCE
    US_ATTRIBUTE_GET_TORQUE
    US_ATTRIBUTE_GET_MASS
    US_ATTRIBUTE_GET_TEMPERATURE
    US_ATTRIBUTE_GET_PRESSURE
    US_ATTRIBUTE_GET_VISCOSITY
    US_ATTRIBUTE_GET_LUMINANCE
    US_ATTRIBUTE_GET_HUMIDITY
    US_ATTRIBUTE_GET_FLOW
    US_ATTRIBUTE_GET_HARDNESS
    US_ATTRIBUTE_GET_ROUGHNESS
    US_ATTRIBUTE_GET_GEOMETRY
    US_ATTRIBUTE_GET_TOPLOGY
    US_ATTRIBUTE_GET_SHAPE
    US_ATTRIBUTE_GET_PATTERN
    US_ATTRIBUTE_GET_MATERIAL
    US_ATTRIBUTE_GET_KINEMATICS
```

H.2 Sorted Interface List

```
US_2D_SENSOR_GET_READING
US_2D_SENSOR_LOAD_ARRAY_PATTERN
US_2D_SENSOR_POST_READING
US_2D_SENSOR_USE_ARRAY_TYPE
US_ABORT
US_ADD_SYMBOLIC_ITEM
US_ADD_SYMBOLIC_ITEM_ATTR
US_ADS_COLLISION_DETECTED
US_ATTRIBUTE_GET_ACCELERATION
US_ATTRIBUTE_GET_FLOW
US_ATTRIBUTE_GET_FORCE
US_ATTRIBUTE_GET_GEOMETRY
US_ATTRIBUTE_GET_HARDNESS
US_ATTRIBUTE_GET_HUMIDITY
US_ATTRIBUTE_GET_JERK
US_ATTRIBUTE_GET_KINEMATICS
US_ATTRIBUTE_GET_LUMINANCE
US_ATTRIBUTE_GET_MASS
US_ATTRIBUTE_GET_MATERIAL
US_ATTRIBUTE_GET_ORIENTATION
US_ATTRIBUTE_GET_PATTERN
US_ATTRIBUTE_GET_POSE
US_ATTRIBUTE_GET_POSITION
US_ATTRIBUTE_GET_PRESSURE
US_ATTRIBUTE_GET_ROUGHNESS
US_ATTRIBUTE_GET_SHAPE
US_ATTRIBUTE_GET_TEMPERATURE
US_ATTRIBUTE_GET_TIME
US_ATTRIBUTE_GET_TOPLOGY
US_ATTRIBUTE_GET_TORQUE
US_ATTRIBUTE_GET_VELOCITY
US_ATTRIBUTE_GET_VISCOSITY
US_ATTRIBUTE_POST_RESPONSE
US_AXIS_SERVO_CLEAR_BRAKES
US_AXIS_SERVO_HOME
US_AXIS_SERVO_JOG
US_AXIS_SERVO_JOG_STOP
US_AXIS_SERVO_LOAD_CYCLE_TIME
US_AXIS_SERVO_LOAD_DAMPING_VALUES
US_AXIS_SERVO_LOAD_DOF
US_AXIS_SERVO_LOAD_GAIN_LIMIT
US_AXIS_SERVO_LOAD_JOINT_LIMIT
US_AXIS_SERVO_LOAD_PID_GAIN
US_AXIS_SERVO_LOAD_VELOCITY_LIMIT
US_AXIS_SERVO_SET_ACCELERATION
US_AXIS_SERVO_SET_BRAKES
US_AXIS_SERVO_SET_CURRENT
US_AXIS_SERVO_SET_FORCES
US_AXIS_SERVO_SET_POSITION
US_AXIS_SERVO_SET_TORQUE
US_AXIS_SERVO_SET_VELOCITY
US_AXIS_SERVO_SET_VOLTAGE
US_AXIS_SERVO_START_GRAVITY_COMPENSATION
US_AXIS_SERVO_STOP_GRAVITY_COMPENSATION
US_AXIS_SERVO_USE_ABS_POSITION_MODE
US_AXIS_SERVO_USE_ABS_VELOCITY_MODE
US_AXIS_SERVO_USE_ANGLE_UNITS
US_AXIS_SERVO_USE_COMPLIANCE
US_AXIS_SERVO_USE_CURRENT
US_AXIS_SERVO_USE_FEEDFORWARD_TORQUE
US_AXIS_SERVO_USE_IMPEDANCE
US_AXIS_SERVO_USE_PID
US_AXIS_SERVO_USE_RADIAN_UNITS
US_AXIS_SERVO_USE_REL_POSITION_MODE
US_AXIS_SERVO_USE_REL_VELOCITY_MODE
US_AXIS_SERVO_USE_STIFFNESS
US_AXIS_SERVO_USE_VOLTAGE
US_BEGIN_BLOCK
US_BEGIN_EVENT
US_BEGIN_FRAMEWORK
US_BEGIN_MACRO
US_BEGIN_PLAN
US_BEGIN_SINGLE_STEP
US_CLEAR_SINGLE_STEP
US_CREATE_FRAMEWORK
US_DELETE_FRAMEWORK
US_DELETE_SYMBOLIC_ITEM
US_DELETE_SYMBOLIC_ITEM_ATTR
US_DISABLE
US_ENABLE
US_END_BLOCK
US_END_EVENT
US_END_FRAMEWORK
US_END_MACRO
US_END_PLAN
US_ERROR_BAD_DATA
US_ERROR_COMMAND_ENTRY
US_ERROR_COMMAND_NOT_IMPLEMENTED
US_ERROR_DUPLICATE_NAME
US_ERROR_LIMIT_EXCEEDED
US_ERROR_NO_DATA_AVAILABLE
US_ERROR_OVER_SPECIFIED
US_ERROR_SAFETY_VIOLATION
US_ERROR_UNDER_SPECIFIED
US_ESTOP
US_FLOW_LOAD_PARAMETERS
US_FLOW_START_FLOOD
US_FLOW_START_MIST
US_FLOW_STOP_FLOOD
US_FLOW_STOP_MIST
US_FT_SENSOR_POST_READING
US_GENERIC_STATUS_REPORT
US_GET_EXT_DATA_VALUE
US_GET_FEATURE
US_GET_LIST
US_GET_OBJECT_ID
US_GET_SELECTION_ID
US_GET_VALUE
US_HALT
US_HOLD
US_IMAGE_ADJUST_FOCUS
US_IMAGE_ADJUST_POSITION
US_IMAGE_COMPUTE_FLOW_MODE
US_IMAGE_COMPUTE_RANGE_MODE
US_IMAGE_COMPUTE_SPATIAL_DERIVATIVES_MODE
US_IMAGE_COMPUTE_TEMPORAL_DERIVATIVES_MODE
US_IMAGE_LOAD_CALIBRATION
US_IMAGE_POST_BYTE_SYMBOLIC_READING
US_IMAGE_POST_FLOW_READING
US_IMAGE_POST_HISTOGRAM_READING
US_IMAGE_POST_PIXEL_MAP_READING
US_IMAGE_POST_RANGE_READING
US_IMAGE_POST_RECOGNITION_READING
US_IMAGE_POST_SPATIAL_DERIVATIVE_READING
US_IMAGE_POST_SPECIFICATION
US_IMAGE_POST_TEMPORAL_DERIVATIVE_READING
US_IMAGE_POST_TRESHOLD_READING
US_IMAGE_POST_XY_CHAR_READING
US_IMAGE_SET_POSITION
US_IMAGE_USE_CENTROID_MODE
```

US_IMAGE_USE_FRAME_GRAB_MODE
US_IMAGE_USE_GRAY_LEVEL_MODE
US_IMAGE_USE_HISTOGRAM_MODE
US_IMAGE_USE_RECOGNITION_MODE
US_IMAGE_USE_SEGMENTATATION_MODE
US_IMAGE_USE_TRESHOLD_MODE
US_INIT
US_LOAD_EXT_PARAMETER
US_LOAD_OBSTACLE
US_LOAD_PATTERN
US_LOAD_STATUS_PERIOD
US_LOAD_STATUS_TYPE
US_MARK_BREAKPOINT
US_MARK_EVENT
US_NEXT_SINGLE_STEP
US_OC_GET_ATTR
US_OC_GET_CALIB
US_OC_SET_ATTR
US_OC_SET_CALIB
US_OK_ATTRIBUTE_QUERY
US_OK_ATTRIBUTE_RESPONSE
US_OK_CREATE_OBJ
US_OK_DELETE_OBJ
US_OK_MODIFY
US_OK_MODIFY_ATTRIBUTE
US_OK_OUTPUT_REGISTERED_OBJ_ID
US_OK_PLAYBACK
US_OK_RECORD
US_OM_CREATE
US_OM_DELETE
US_OM_MODIFY
US_PAUSE
US_PIO_BIT_READ
US_PIO_BIT_SET
US_PIO_CONTROL_WRITE
US_PIO_DATA_READ
US_PIO_DATA_WRITE
US_PIO_DISABLE
US_PIO_ENABLE
US_PIO_LOAD_SCALE
US_PIO_POST_DATA
US_PIO_SET_MODE
US_PIO_TOGGLE_BIT
US_POST_EXT_DATA_VALUE
US_POST_ID
US_POST_LIST
US_POST_SELECTION_ID
US_POST_VALUE
US_PTPS_INPUT_REQUEST
US_PTPS_INTERP_HALT_PLAN
US_PTPS_INTERP_RUN_PLAN
US_PTPS_OUTPUT_ENABLE_SUBSYSTEM
US_PTPS_SELECT_AGENT
US_PTPS_SELECT_SENSOR
US_RESET
US_RESUME
US_SCALAR_SENSOR_POST_READING
US_SENSOR_GET_ATTRIBUTES_READING
US_SENSOR_GET_READING
US_SENSOR_LOAD_FILTER
US_SENSOR_LOAD_FREQUENCY
US_SENSOR_LOAD_SAMPLING_SPEED
US_SENSOR_LOAD_TRANSFORM
US_SENSOR_USE_MEASUREMENT_UNITS
US_SET_EXT_DATA_VALUE
US_SET_SYMBOLIC_ITEM_ATTR
US_SHUTDOWN
US_SPINDLE_LOAD_SPEED
US_SPINDLE_LOCK_Z
US_SPINDLE_ORIENT
US_SPINDLE_RETRACT
US_SPINDLE_RETRACT_TRAVERSE
US_SPINDLE_START_TURNING
US_SPINDLE_STOP_TURNING
US_SPINDLE_USE_FORCE
US_SPINDLE_USE_NO_FORCE
US_START
US_STARTUP
US_START_FILTER
US_START_TRANSFORM
US_STOP
US_STOP_FILTER
US_STOP_TRANSFORM
US_TDS_EXECUTE_PROGRAM
US_TDS_LOAD_ORIGIN
US_TDS_LOAD_RATE_DEFAULTS
US_TDS_LOAD_REFERENCE_UNITS
US_TDS_LOAD_SELECTIONS
US_TDS_LOAD_SENSING_DEFAULTS
US_TDS_LOAD_USER
US_TDS_SELECT_OPERATION
US_TDS_SELECT_OPMODE
US_TDS_SELECT_PROGRAM
US_TK_DEFINE_FRAMEWORK
US_TK_MACRO_CREATE
US_TK_MACRO_DELETE
US_TK_MACRO_MODIFY
US_TLC_ADJUST_AXIS
US_TLC_ARC_FEED
US_TLC_CHANGE_TOOL
US_TLC_GOAL_SEGMENT
US_TLC_INCR_ACCELERATION
US_TLC_INCR_VELOCITY
US_TLC_LOAD_ACCELERATION
US_TLC_LOAD_BASE_PARAMETERS
US_TLC_LOAD_CONTACT_FORCES
US_TLC_LOAD_CONTACT_FORCE_LIMIT
US_TLC_LOAD_CONTACT_TORQUE_LIMIT
US_TLC_LOAD_CYCLE_TIME
US_TLC_LOAD_DELTA
US_TLC_LOAD_DOF
US_TLC_LOAD_FEED_RATE
US_TLC_LOAD_JERK
US_TLC_LOAD_JOINT_LIMIT
US_TLC_LOAD_KINEMATIC_RING_POSITIONING_MODE
US_TLC_LOAD_LENGTH_UNITS
US_TLC_LOAD_NEIGHBORHOOD
US_TLC_LOAD_OBJECT
US_TLC_LOAD_OBJECT_BASE
US_TLC_LOAD_OBJECT_OFFSET
US_TLC_LOAD_OBSTACLE_VOLUME
US_TLC_LOAD_PROXIMITY
US_TLC_LOAD_RELATIVE_POSITIONING
US_TLC_LOAD_REPRESENTATION_UNITS
US_TLC_LOAD_SEGMENT_TIME
US_TLC_LOAD_SENSOR_FUSION_ORIENT_LIMIT
US_TLC_LOAD_SENSOR_FUSION_POS_LIMIT
US_TLC_LOAD_TERMINATION_CONDITION
US_TLC_LOAD_TOOL_PARAMETERS
US_TLC_LOAD_TRAVERSE_RATE
US_TLC_NURBS_CONTROL_POINT
US_TLC_NURBS_FEED

US_TLC_NURBS_KNOT_VECTOR
US_TLC_PARAMETRIC_2D_CURVE_FEED
US_TLC_PARAMETRIC_3D_CURVE_FEED
US_TLC_SELECT_PLANE
US_TLC_SET_GOAL_POSITION
US_TLC_START_AUTOMATIC_MOTION
US_TLC_START_COMPLIANT_MOTION
US_TLC_START_CUTTER_RADIUS_COMPENSATION
US_TLC_START_FINE_MOTION
US_TLC_START_FORCE_POSITIONING_MODE
US_TLC_START_GUARDED_MOTION
US_TLC_START_MANUAL_MOTION
US_TLC_START_MOVE_UNTIL_MOTION
US_TLC_START_STANDOFF_DISTANCE
US_TLC_START_TRAVERSE_MOTION
US_TLC_STOP_AUTOMATIC_MOTION
US_TLC_STOP_COMPLIANT_MOTION
US_TLC_STOP_CUTTER_RADIUS_COMPENSATION
US_TLC_STOP_FINE_MOTION
US_TLC_STOP_FORCE_POSITIONING_MODE
US_TLC_STOP_GUARDED_MOTION
US_TLC_STOP_MANUAL_MOTION
US_TLC_STOP_MOVE_UNTIL_MOTION
US_TLC_STOP_STANDOFF_DISTANCE
US_TLC_STOP_TRAVERSE_MOTION
US_TLC_STRAIGHT_FEED
US_TLC_STRAIGHT_TRAVERSE
US_TLC_TELEOP_FORCE_REFLECTION_UPDATE
US_TLC_UPDATE_SENSOR_FUSION
US_TLC_USE_ABSOLUTE_POSITIONING_MODE
US_TLC_USE_CARTESIAN_REFERENCE_FRAME
US_TLC_USE_CUTTER_RADIUS_COMPENSATION
US_TLC_USE_JOINT_REFERENCE_FRAME
US_TLC_USE_KINEMATIC_RING_POSITIONING_MODE
US_TLC_USE_MODIFIED_TOOL_LENGTH_OFFSETS
US_TLC_USE_NORMAL_TOOL_LENGTH_OFFSETS
US_TLC_USE_NO_TOOL_LENGTH_OFFSETS
US_TLC_USE_RELATIVE_POSITIONING_MODE
US_TLC_USE_REPRESENTATION_UNITS
US_TLC_USE_TOOL_TIP_COORDINATE_FRAME
US_TLC_USE_WRIST_COORDINATE_FRAME
US_TLC_ZERO_PROGRAM_ORIGIN
US_TLC_ZERO_RELATIVE_POSITIONING
US_TPS_CONTACT_MOTION
US_TPS_ENABLE
US_TPS_FREESPACE_MOTION
US_TPS_GUARDED_MOTION
US_TPS_MARK_EVENT
US_TPS_SELECT_FEATURE
US_TPS_SELECT_MATERIAL
US_TPS_SELECT_TOOL
US_TPS_SET_SUPERVISORY_MODE
US_TRD_ADD_ELEMENT
US_TRD_DELETE
US_TRD_DELETE_ITEM
US_TRD_ERASE
US_TRD_FIND
US_TRD_MODIFY
US_TRD_NAME_ITEM
US_TRD_NEXT
US_TRD_OPEN
US_TRD_PREVIOUS
US_TRD_RECORD
US_TRD_RECORD_OFF
US_TRD_RECORD_ON
US_TRD_SET_CARTESIAN_MODE
US_TRD_SET_JOINT_MODE
US_USE_AXIS_MASK
US_USE_EXT_ALGORITHM
US_USE_FEATURE
US_USE_MACRO
US_USE_OBJECT
US_USE_PLAN
US_USE_SELECTION
US_VECTOR_SENSOR_GET_READING
US_VECTOR_SENSOR_POST_READING
US_ZERO

H.3 Interface Source Listings

H.3.1 Disclaimer

```
//====================== utap_disclaimer.h =================================
//
//
// Unified Telerobotic Architecture Project (UTAP)
// Interface Definitions
// Release:  1.0
// Revision  0.0
// Release Date: 24-May-1994
//
//

#define UTAP_VERSION 1.0

//
// DISCLAIMER:
//
// This software was produced by the National Institute of Standards and
// Technology (NIST), an agency of the U.S. government, and by statute is
// not subject to copyright in the United States.  Recipients of this
// software assume all responsibility associated with its operation,
// modification, maintenance, and subsequent redistribution.
//
//


/*
   Modification History:

   06/18/94  jlm  Public Release of Messages

   05/23/94  jlm  Modified definitions for greater consistency.

   04/24/94  jlm  Created

*/
```

H.3.2 Generic Definitions

```
//====================== generic_types.h ============================
//
//    FILE : generic_types.h
//
//    PURPOSE : This file contains a list of domain-independent types
//
//    DATE    : Novemeber 17, 1993
//
//

#ifndef UTAP_GENERIC_DEFINITIONS
#define UTAP_GENERIC_DEFINITIONS


//
// MODE_DIRECTIVES - class to define enumerated set of process modes
```

```
//
class MODE_DIRECTIVE {
    enum  {
        abort         = 0x10001,
        halt          = 0x10002,
        pause         = 0x10004,
        resume        = 0x10008,
        reset         = 0x10011,
        estop         = 0x10012,
        report        = 0x10014,
        start         = 0x10018,
        shutdown      = 0x10020,
        hold          = 0x10021,
        reinitialize = 0x10022,
    } ;
};

//
// GENERIC_DIRECTIVES - class of enumerated set of
//
class GENERIC_DIRECTIVES : public MODE_DIRECTIVE {
    enum
      {
          no_change          = 0x0000,            // use same parameter
          no_selection       = 0x0001,            // parameter not required
          delegate_selection = 0x0002,            // let subordinate decide parameter
          no_op              = 0x0004             // slot for commandless mode directive
      } ;
};

//
// LOGICAL TYPE - enumerated list of logical states
//
typedef enum {
#if !defined(TRUE)
                TRUE = 1,
#endif
#if !defined(FALSE)
                FALSE = 0 ,
#endif
                ALL  = -1,                        // good for bitmask
            }                                    LOGICAL;

//
// USER_TYPE
//
typedef enum USER_TYPE {
        ATTENDANT       = 1,
        OPERATOR        = 2,
        PROGRAMMER      = 3,
        MANAGER         = 4,
        MAINTENANCE     = 5,
        SYSTEMS         = 6,
        ROOT            = 8,
} USER_TYPE;

//
// MODE_STATE
//
typedef enum {
    calibration        = 0x40001,
    diagnostic         = 0x40002,
    maintenance        = 0x40004,
    normal_operation   = 0x40008,
    safe               = 0x40010,
    shutdown           = 0x40012,
    initialize         = 0x40014,
    training           = 0x40018,
    teleoperation      = 0x40020,
    shared             = 0x40020
  }                                                  MODE_STATE;


//
// RESULT_TYPE - enumerated set of result possibilities
//
typedef enum {
  failed            = -1,
  incomplete        =  0,
  succeeded         =  1,
  partial_sucess    =  2
  //  exception     = -2,
  // exception is different kind of failure
}                                                    RESULT_TYPE;

//
// STATE_TYPE - enumerated set of
//
typedef enum {
    finished,                                      //or is done better ? , see result_type
    ready,
    halted,
    suspended,
    aborted,
    resetting,
    exception,
    executing                                      // same as running
    /* executing_forward, /* future */
    /* executing_backward,  /* future */
}                                                    STATE_TYPE;

//
// STATUS_TYPE - synonym of STATE_TYPE
//
typedef STATE_TYPE                                   STATUS_TYPE;

//
// REQUEST_TYPE - enumerate set of request states
//
typedef enum {
    request_started,
    request_pending,
    request_complete,
    request_blocked,
    request_failed,
    request_aborted
  }                                                  REQUEST_TYPE;


//
// TIME - get POSIX definition
//
typedef double TIME;

//
// TIMELINE - struct definition of time frame
//
struct TIMELINE {
    TIME            duration;                      //  how long to take
    TIME            earliest_start;                //  earliest to start
    TIME            latest_start;                  //  latest to start
    TIME            earliest_completion;           //  earliest to finish
    TIME            latest_completion;          //  latest to finish

};

//
// SEVERITY_TYPE - enumerated definition of severity types
//
typedef enum {
    fatal,
    severe,
    warning,
    informative
  }                                             SEVERITY_TYPE;

//
// POSITIONING_TYPE - enumerated definition of positioning types
//
typedef enum {
    absolute,
    incremental,
    jog,
    relative,
  }                                             POSITIONING_TYPE;


#endif
```

H.3.3 Classification

```
//======================= utap_classification.h ===========================
//    MODULES ACRONYMS:
//    ================
//
//    TDS - task description and supervision
//    TPS - task program  sequencer
//    TPS - parent program  sequencer
//    TLC - task level control
//    DC  - device control
//
//    OI  - operator interface
//    OK  - object knowledge
//    TK  - task knowledge
//    TD  - trajectory description
//    SGD - status graphics displays
//    SS  - subsystem simulators
//    AD  - analysis and diagnosis
//    VS  - virtual sensor
//    DB  - data base
//    SC  - sensor control
//    AC  - axis servo control

//
// UTAP Classification Typing
//

#ifndef UTAP_CLASSIFICATION
#define UTAP_CLASSIFICATION

enum { _JOYSTICK,
       _F_R_JOYSTICK,
       _PENDANT,
       _PANEL,
       _WINDOWS,
   }                                             US_OI_MODULE_TYPES;

enum { _TEACH        = 0x01,
       _SCRIPTED     = 0x02,
       _PROGRAMMABLE = 0x04,
   }                                             US_TD_MODULE_TYPES;

enum { _MANIPULATION = 0x01,
       _NAVIGATION   = 0x02,
       _TOOLING      = 0x04,
       _MACHINING    = 0x08,
       // obviously more
   }                                             US_TPS_MODULE_TYPES;

enum { _PICK_PLACE   = 0x01,
       _DEXTROUS     = 0x02,
       //???
   }                                             US_TPS_MANIPULATION_TYPES;

enum { _TELEOP       = 0x01,
       _GUIDED       = 0x02,
       _AUTONOMOUS   = 0x04,
       //???
   }                                             US_TPS_NAVIGATION_TYPES;

enum { _VERTICAL     = 0x01,
       _HORIZONTAL   = 0x02,
       _TURNING      = 0x04,
       _EDM          = 0x08,
       //???
   }                                             US_TPS_MACHINING_TYPES;

enum { _CONTACT      = 0x01,
       _NONCONTACT   = 0x02,
   }                                             US_TPS_TOOLING_TYPES;

enum { _DENAVIT_HARTENBURG = 0x01,
       _SCARA              = 0x02,
       _GANTRY             = 0x03,
       _STEWART_PLATFORM   = 0x04,
       // obviously more
   }                                             US_ROBOT_TYPES;


enum { _SPRAY              = 0x01,
       _FINISH             = 0x02,
   }                                             US_TOOL_TYPES;

enum { _FTS                =  0x1,
       _IMAGE              =  0x2,
       _PROBE              =  0x3,
       _SWITCH             =  0x4,
       _RANGE              =  0x5,
   }                                             US_SENSOR_TYPES;


#endif
```

H.3.4 Protocol

```
//====================== utap_protocol.h ================================
//
// UTAP Protocol Typing
//


struct MsgTransmitHeader {

    int byte_order;                                 // big/little endian?
    int command_num;                                // increment with every new command
    // any others??
};

struct MsgAckHeader {

    int byte_order;                                 // big/little endian?
    int echo_message_num;                           // acknowledge receipt
    int health;                                     // health of device - mimics Lords Sensor
    // any others??
};


MODE_DIRECTIVE      mode;                           // combine mode x command


typedef enum {
    read_only,
    write_only,
    read_write
  }                                        ACCESS_TYPE;

typedef int CHANNEL__;

typedef enum {
    SEND,
    RECEIVE
  }                                        COMMUNICATION_DIRECTION_TYPE;

typedef enum {  local_procedure_call,
                remote_procedure_call,
                sw_interrupt,
                event,
                signal,
                MMS,
                network_comm,
                shared_memory,
                message_queue,
                mailbox,
                SP_50,
                SERCOS,
                CAN,
            }                              CONNECTION_TYPE;
```

H.3.5 Information Model

```
//======================= utap_data_defs.h ===============================
#ifndef UTAP_DATA_DEFS
#define UTAP_DATA_DEFS
#include "generic_defs.h"

typedef
enum {  X_AXIS          =     0x01,
        Y_AXIS          =     0x02,
        Z_AXIS          =     0x04,
        POSITION_AXES   =     0x07,

        ROLL_AXIS       =     0x08,
        PITCH_AXIS      =     0x10,
        YAW_AXIS        =     0x20,
        ORIENTATION_AXES =    0x38,


        JOINT1_AXIS     =     0x01,
        JOINT2_AXIS     =     0x02,
        JOINT3_AXIS     =     0x04,
        JOINT4_AXIS     =     0x08,
        JOINT5_AXIS     =     0x10,
        JOINT6_AXIS     =     0x20,
        JOINT7_AXIS     =     0x40,
        JOINT8_AXIS     =     0x80,
        JOINT9_AXIS     =    0x100,
        JOINT10_AXIS    =    0x200,


        // Modifiers
        ELBOW           =     0x1000,
        WRIST           =     0x2000,
        TOOLTIP         =     0x4000,

    } AxisMask;


typedef
     enum { unitless_u    =  0x00,
            meters_u     =   (1L<<1),
            grams_u      =   (1L<<2),
            liters_u     =   (1L<<3),
            seconds_u    =   (1L<<4),
            radians_u    =   (1L<<5),
            angles_u     =   (1L<<6),
            newtons_u    =   (1L<<7),
            celsius_u    =   (1L<<8),
            pascal_u     =   (1L<<9),
            lumin_u      =   (1L<<10),
            psi_u        =   (1L<<11),
            rpm_u        =   (1L<<12),
            Hz_u         =   (1L<<13),
            cardinal_u   =   (1L<<14),
            updown_u     =   (0x1L << 33),

            // Non-SI Modifier
            nano_u       =   (1L<<20),
            micro_u      =   (1L<<21),
            milli_u      =   (1L<<22),
            kilo_u       =   (1L<<23),
            nonSI_modifier =  (0xFL << 20),

            // Non-SI altogether
            inches_u     =   (1L<<30),
            feet_u       =   (1L<<31),
            pounds_u     =   (1L<<32),
            English_units =  (0xFL << 30),

        } Measurement_units_type;

typedef
    enum {
        char_t              =   0x0100001,
        short_t             =   0x0100002,
        int_t               =   0x0100003,
        long_t              =   0x0100004,
        u_char_t            =   0x0100005,
        u_short_t           =   0x0100006,
        u_int_t             =   0x0100007,
        u_long_t            =   0x0100008,
        float_t             =   0x0100009,
        double_t            =   0x010000A,
        array_t             =   0x0200000,
        ptr_t               =   0x0400000,

        cartesian_t         =   0x1000000,
        spherical_t         =   0x2000000,
        cylindrical_t       =   0x3000000,
        H_matrice_t         =   0x4000000,        // naop homogeneous transform matrix
        transform_t         =   0x4000000,        // ibid
        Euler_t             =   0x5000000,        // Euler Angles
        ZYXEuler_t          =   0x5000000,        // ZYX Euler Angles
        ZYZEuler_t          =   0x6000000,        // ZYZ Euler Angles
        Quaternion_t        =   0x7000000,        // Quaternian Angles
        Equiv_Angle_Axis_t  =   0x8000000,        // Equivalent Angle Axis
        RPY_t               =   0x9000000,        // Roll Pitch Yaw

        geometry_t          =   0x10000000,
        topology_t          =   0x20000000,
        material_t          =   0x30000000,
        shape_t             =   0x40000000,
        pattern_t           =   0x50000000,
        kinematics_t        =   0x60000000,

        bitmask_t           =   0x100000000,

    } Representation_units_type;

//
// Object type
//
struct Object_type  {
    int  id;
    enum { _location_ = 0x80000,
           _part_     = 0x80001,
           _simple_   = 0x80002,
           _robot_    = 0x80003,
           _tool_     = 0x80004,
           _list_     = 0x80005,
           _module_   = 0x80006,
       } type;
};

//
// Attribute Types - Enumeration
//
typedef enum {
    _object_name_                =   0x0000001,
    _attribute_name_             =   0x0000002,
    _material_name_              =   0x0000004,
    _time_                       =   0x0000008,
    _position_                   =   0x0000040,
    _orientation_                =   0x0000080,
    _pose_                       =   0x0000100,
    _velocity_                   =   0x0000200,
    _acceleration_               =   0x0000400,
    _jerk_                       =   0x0000800,
    _force_                      =   0x0001000,
    _torque_                     =   0x0002000,
    _temperature_                =   0x0004000,
    _pressure_                  =  0x0008000,
    _viscosity_                 =  0x0010000,
    _luminance_                 =  0x0020000,
    _humidity_                  =  0x0040000,
    _flow_                      =  0x0080000,
    _hardness_                  =  0x0100000,
    _roughness_                 =  0x0200000,
    _mass_                      =  0x0400000,

    _geometry_                  =  0x01000000,
    _topology_                  =  0x02000000,
    _shape_                     =  0x04000000,
    _pattern_                   =  0x08000000,
    _material_                  =  0x10000000,
    _kinematics_                =  0x20000000,

    // where does this info belong?
    _link_length_               = 0x100000000,
    _link_twist_                = 0x200000000,
    _link_offset_               = 0x400000000,
    _link_mass_                 = 0x800000000,
    _link_encoder_ticks_        = 0x1000000000,

#if 0
    // Not supported from hereon in
       _elasticity_
       _spring_constant_,
       _illumination_,
       _pitch_,
       _loudness_,
       _intensity_,
       _amplitude_,
       _frequency
       _count_,
       _period_,
       _phase_,
#endif
} Attribute_t;

//
// State Modifier of Attribute
//
typedef enum {
    all       =  -1,
    translational = 0x00001,
    rotational    = 0x00002,

    // sensing modifiers - more get oriented
    actual        = 0x00100,
    desired       = 0x00200,
    max           = 0x00400,
    min           = 0x00800,
    last          = 0x01000,

    // positiong modifier - more set oriented
    // absolute      = 0x02000,
    // relative      = 0x04000,
    // incremental   = 0x08000,
    // jog           = 0x10000,

} Modifier_t;

//
//  Generic Attribute Data Storage
//
#include <sys/types.h>
```

```
struct generic_value_a {
  public:
    union {
        char    c;
        short   s;
        int     i;
        long    l;
        u_char  uc;
        u_short us;
        u_int   ui;
        u_long  ul;
        float   f;
        double  d;
        void  * heap;                        // variable data follow in heap format
    } value, min, max;
};

//
// ROUTE - struct to define read or get query routing destination
//
struct ROUTE  {
    enum { _STATUS        = 1,                // post response to questioner
           _WRITE_TO_OK   = 2,                // posting response values to cental obj knowl
           _READ_FROM_OK  = 4,                // get next values from obj knowl
           _DELTA_OFFSET  = 8,                // use data as delta offset
           _ALTER         = 8,                // to alter cmd dx,dy,dz,rx,ry,rz
    } type;                                   // Bitmask to indicate destination for response
    int times;                                // 0 means continuous, 1= one read,...
    TIME update_period;                       // frequency of update
    int offset;                               // optional delta offset position
};

//
// General Purpose
//

#define US_POST_ID 50
struct us_post_id_msg_t {
    int msgid;
    int id;
};

#define US_GET_OBJECT_ID 51
struct us_get_object_id_msg_t {
    int msgid;
    char name[128];
};

#define US_USE_OBJECT 52
struct us_use_object__msg_t {
    int msgid;
    int id;
};

#define US_GET_FEATURE 53
struct us_get_feature__msg_t {
    int msgid;
    char name[128];
    ROUTE r;
};

#define US_USE_FEATURE 54
struct us_use_feature_msg_t {
    int msgid;
    int id;
};
```

```
#define US_GET_VALUE 55
struct us_get_value_msg_t {
    int msgid;
    ROUTE r;
    Attribute_t items;
    Modifier_t modifiers;
};

#define US_POST_VALUE 56
struct us_post_value_msg_t {
    int msgid;
    int id;
    Attribute_t item;
    Modifier_t modifier;
    Representation_units_type rep;
    Measurement_units_type units;
    generic_value_a value;
};

#define US_GET_LIST 57
struct us_get_list_msg_t {
    int msgid;
    ROUTE r;
    Attribute_t items;
    Modifier_t modifiers;
};

#define US_POST_LIST 58
struct us_post_list_msg_t {
    int msgid;
    Attribute_t items;
    Modifier_t modifiers;
    generic_value_a *values;
};

//
//
// Object Knowledge Specific Attribute Messages
//
//
#define US_ATTRIBUTE_POST_RESPONSE 1600
struct us_attribute_post_response_msg_t {
    int msgid;
    int id;
    Attribute_t item;
    Modifier_t modifier;
    int size;
    Representation_units_type rep;
    Measurement_units_type units;
    generic_value_a value;
};

#define US_ATTRIBUTE_GET_TIME 1601
struct us_attribute_get_time_msg_t {
    int msgid;
    int id;
    ROUTE r;
    Modifier_t modifier;
    Measurement_units_type desired_units;
};


#define US_ATTRIBUTE_GET_POSITION 1602
struct us_attribute_get_position_msg_t {
    int msgid;
    int id;
    ROUTE r;
    Modifier_t modifier;
    Representation_units_type  rep = double_t;
    Measurement_units_type units = meters_u;
};

#define US_ATTRIBUTE_GET_ORIENTATION 1603
struct us_attribute_get_orientation_msg_t {
    int msgid;
    int id;
    ROUTE r;
    Modifier_t  modifier;
    Measurement_units_type desired_units = radians_u;
};

#define US_ATTRIBUTE_GET_POSE 1604
struct us_attribute_get_pose_msg_t {
    int msgid;
    int id;
    ROUTE r;
    Modifier_t modifier;
    Measurement_units_type desired_pos_units;
    Measurement_units_type desired_rot_units;
};

#define US_ATTRIBUTE_GET_VELOCITY 1605
struct us_attribute_get_velocity_msg_t {
    int msgid;
    int id;
    ROUTE r;
    Modifier_t modifier;
    Measurement_units_type desired_units = meters_u;
};

#define US_ATTRIBUTE_GET_ACCELERATION 1606
struct us_attribute_get_acceleration_msg_t {
    int msgid;
    int id;
    ROUTE r;
    Modifier_t  modifier;
    enum { time_to_accel_u,
           meters_per_sec_per_sec
       } desired_units = meters_per_sec_per_sec ;
};

#define US_ATTRIBUTE_GET_JERK 1607
struct us_attribute_get_jerk_msg_t {
    int msgid;
    int id;
    Modifier_t modifier;
    enum { meters_per_sec_per_sec } units;
};

#define US_ATTRIBUTE_GET_FORCE 1608
struct us_attribute_get_force_msg_t {
    int msgid;
    int id;
    ROUTE r;
    Modifier_t modifier;
    Measurement_units_type desired_units = newtons_u;
};

#define US_ATTRIBUTE_GET_TORQUE 1609
struct us_attribute_get_torque_msg_t {
    int msgid;
    int id;
    ROUTE r;
    Modifier_t modifier;
    enum { newtons_per_meter } desired_units;
};

#define US_ATTRIBUTE_GET_MASS 1610
struct us_attribute_get_mass_msg_t {
    int msgid;
    int id;
    ROUTE r;
    Modifier_t modifier;
    int size;
    Measurement_units_type desired_units = grams_u;
};

#define US_ATTRIBUTE_GET_TEMPERATURE 1611
struct us_attribute_get_temperature_msg_t {
    int msgid;
    int id;
    ROUTE r;
    Modifier_t  modifier;
    Measurement_units_type desired_units = celsius_u;
};


#define US_ATTRIBUTE_GET_PRESSURE 1612
struct us_attribute_get_pressure_msg_t {
    int msgid;
    int id;
    Modifier_t modifier;
    Measurement_units_type desired_units = pascal_u;
};

#define US_ATTRIBUTE_GET_VISCOSITY 1613
struct us_attribute_get_viscosity_msg_t {
    int msgid;
    int id;
    ROUTE r;
    Modifier_t modifier;
    enum { mPa_per_second} desired_units;
};

#define US_ATTRIBUTE_GET_LUMINANCE 1614
struct us_attribute_get_luminance_msg_t {
    int msgid;
    int id;
    ROUTE r;
    Modifier_t modifier;
    Measurement_units_type desired_units = lumin_u;
};

#define US_ATTRIBUTE_GET_HUMIDITY 1615
struct us_attribute_get_humidity_msg_t {
    int msgid;
    int id;
    ROUTE r;
    Modifier_t modifier;
    enum {grams_per_meter_cubed } desired_units;
};

#define US_ATTRIBUTE_GET_FLOW 1616
struct us_attribute_get_flow_msg_t {
    int msgid;
    int id;
    ROUTE r;
    Modifier_t modifier;
    Measurement_units_type desired_units;
};

#define US_ATTRIBUTE_GET_HARDNESS 1617
struct us_attribute_get_hardness_msg_t {
    int msgid;
    int id;
    ROUTE r;
    Modifier_t modifier;
    Measurement_units_type desired_units;
};

#define US_ATTRIBUTE_GET_ROUGHNESS 1618
struct us_attribute_get_roughness_msg_t {
    int msgid;
    ROUTE r;
    Modifier_t modifier;
    Measurement_units_type desired_units;
};

#define US_ATTRIBUTE_GET_GEOMETRY 1619
struct us_attribute_get_geometry_msg_t {
    int msgid;
    int id;
    ROUTE r;
    Modifier_t modifier;
    Measurement_units_type desired_units;
};

#define US_ATTRIBUTE_GET_TOPLOGY 1620
struct us_attribute_get_topology_msg_t {
    int msgid;
    int id;
    ROUTE r;
    Modifier_t modifier;
};

#define US_ATTRIBUTE_GET_SHAPE 1621
struct us_attribute_get_shape_msg_t {
    int msgid;
    int id;
    ROUTE r;
    Modifier_t modifier;
};

#define US_ATTRIBUTE_GET_PATTERN 1622
struct us_attribute_get_pattern_msg_t {
    int msgid;
    ROUTE r;
    Modifier_t modifier;
};

#define US_ATTRIBUTE_GET_MATERIAL 1623
struct us_attribute_get_material_t {
    int msgid;
    ROUTE r;
    Modifier_t modifier;
};

#define US_ATTRIBUTE_GET_KINEMATICS 1624
struct us_attribute_get_kinematics_t {
    int msgid;
    ROUTE r;
    Modifier_t modifier;
};
```

```
#endif
```

H.3.6 Interfaces

```
#ifndef UTAP_INTERFACE_DEFINITIONS
#define UTAP_INTERFACE_DEFINITIONS

#include "generic_defs.h"
#include "utap_info_model.h"
#include "utap_data_defs.h"

// These types must be defined - there are stubbed out for now
#include "undefined_types.h"

/**
   This header file defines the interfaces for communication between
   modules in the Generic C5 Architecture.
   Generic Telerobotic Architecture for C-5 Industrial Processes
   contains  modules of which the following have acronyms:

   MODULES NAMING/ACRONYMGS:
   REMOTE:
     RSC   - robot servo control
     TOOL  - tool control
     SENSOR- sensor control
     PIO   - programmable io
     TLC   - task level control
     CLC   - closed loop control
     DB    - data base is part of TLC & CLC
     VS    - virtual sensor
  LOCAL:
     TDS   - task description and supervision
     TK    - task knowledge
     TRD   - trajectory description
     PTPS  - parent task program  sequencer
     TPS   - task program  sequencer
     OI    - operator interface
     OK    - object knowledge
     OC    - object calibration
     OM    - object modeling
     SGD   - status graphics displays
     SS    - subsystem simulators
     ADS   - analysis and diagnosis
  */
/**
 A little table of contents:

  GENERIC:        101  -   199
  ERROR:         -100  -  -200
  ROBOT_SERVO:    200  -   299
  TOOL:           300  -   399
  SENSOR:         400  -   499
  PIO:            500  -   599
  TLC:            600  -   699
  DB:             700  -   799
  VS:             800  -   899

  TDS:           1000  -  1099
  TK:            1100  -  1199
  TRD:           1200  -  1299
  PTPS:          1300  -  1399
  TPS:           1400  -  1499
  OI:            1500  -  1599
  OK            1600  -  1699
  OC            1700  -  1799
  OM            1800  -  1899
  SGD:          1900  -  1999
  ADS:          2000  -  2099
  SS:           2100  -  2199
  */
//
//
// Generic US messages to any Module
// To be verified against RIA Standard R15-06-1992
//
#define GENERIC 100

// Hardware State/Mode Control
#define US_STARTUP 101                                //hardware powered up into safe state
struct us_startup_msg_t {
    int msgid;
};

#define US_SHUTDOWN 102
struct us_shutdown_msg_t {
    int msgid;
};

#define US_RESET 103
struct us_reset_msg_t {
    int msgid;
    enum { HW = 1,
           SW = 2,
       } type;
    long  mask;                                       // bit-map of units to reset
};

#define US_ENABLE 104                                 // sensor/effector(s) turned on
struct us_enable_msg_t {
  int msgid;
  int axis;
};

#define US_DISABLE 105                                // sensor/effector(s) turned off
struct  us_disable_msg_t {
  int msgid;
  int axis;
};

#define US_ESTOP 106                                  // emergency sensor/effector off
struct us_estop_msg_t {
    int msgid;
};

// Software State/Mode Control
#define US_START 107
struct us_start_msg_t {
    int msgid;
};

#define US_STOP 108
struct us_stop_msg_t {
    int msgid;
};

#define US_ABORT 109
struct us_abort_msg_t {
  int msgid;
};
```

```
#define US_HALT  110
struct us_halt_msg_t {
  int msgid;
};

#define US_INIT 111
struct us_init_msg_t {
    int msgid;
};

#define US_HOLD 112
struct us_hold_msg_t {
    int msgid;
};

#define US_PAUSE 113
struct us_pause_msg_t {
  int msgid;
};

#define US_RESUME 114
struct us_resume_msg_t {
  int msgid;
};

#define US_ZERO 115
struct us_zero_msg_t {
    int msgid;
    long mask;                                  // bit-map of units to zero
};

#define US_BEGIN_SINGLE_STEP 116
struct us_begin_single_step_msg_t {
    int msgid;
    // require more explicit info here
};

#define US_NEXT_SINGLE_STEP 117
struct us_next_single_step_msg_t {
    int msgid;
    // require more explicit info here
};

#define US_CLEAR_SINGLE_STEP 118
struct us_clear_single_step_msg_t {
    int msgid;
};

// Interface Programming Constructs
#define US_BEGIN_BLOCK 119
struct us_begin_block_msg_t {
    int msgid;
};

#define US_END_BLOCK 120
struct us_end_block_msg_t {
    int msgid;
};

#define US_BEGIN_PLAN  121
struct us_begin_plan_msg_t {
    int msgid;
    char name[128];
};
```

```
#define US_END_PLAN 122
struct us_end_plan_msg_t {
    int msgid;
};

#define US_USE_PLAN 123
struct us_use_plan_msg_t {
    int msgid;
};

#define US_BEGIN_MACRO 124
struct us_begin_macro_msg_t {
    int msgid;
    char name[128];
};

#define US_END_MACRO 125
struct us_end_macro_msg_t {
    int msgid;
};

#define US_USE_MACRO 126
struct us_execute_macro_msg_t {
    int msgid;
    char name[128];
};

#define US_BEGIN_EVENT 127
struct us_begin_event_msg_t {
    int msgid;
    char name[128];
    enum { _FROM_START        = 1,
           _FROM_END          = 2,
           _AT_TIME           = 3,
           _AT_MARK           = 4,
           _WITH_EXCEPTION    = 5,
       } type;
    TIME t;
    // require step number in block?
};

#define US_END_EVENT 128
struct us_end_event_msg_t {
    int msgid;
};

#define US_MARK_BREAKPOINT 129
struct us_mark_breakpoint_msg_t {
    int msgid;                                     // software pause

};

#define US_MARK_EVENT 130
struct us_mark_event_msg_t {
    int msgid;
    char name[128];                                // place event marker
};

#define US_GET_SELECTION_ID 131
struct us_get_selection_id_msg_t {
    int msgid;
    char name[128];                                // if symbolic get device or module numeric id
};

#define US_POST_SELECTION_ID 132
struct us_post_selection_id_msg_t {
    int msgid;
    int id;
};

#define US_USE_SELECTION 133
struct us_use_selection_msg_t {
    int msgid;
    int id;                                         // which device or module, start with 1
};

#define US_USE_AXIS_MASK 134
struct us_use_axis_mask_msg_t {
    int msgid;
    AxisMask axis;
};

// New Message EXTension Facility
#define US_USE_EXT_ALGORITHM 135
struct us_use_ext_algorithm_msg_t {
    int msgid;
    int slot;                                       // slot holder
};

#define US_LOAD_EXT_PARAMETER 136
struct us_load_ext_parameter_msg_t {
    int msgid;
    int slot;                                       // slot id
};

#define US_GET_EXT_DATA_VALUE 137
struct us_get_ext_data_value_msg_t {
    int msgid;
    int slot;                                       // slot id
    ROUTE r;
};

#define US_POST_EXT_DATA_VALUE 138
struct us_post_ext_data_value_msg_t {
    int msgid;
    int slot;                                       // slot id
    void * data;                                    // pointer into heap
};

#define US_SET_EXT_DATA_VALUE 139
struct us_set_ext_data_value_msg_t {
    int msgid;
    int slot;                                       // slot id
    void * data;                                    // pointer into heap
};

// Status
#define US_LOAD_STATUS_TYPE 140
struct us_load_status_msg_t {
    int msgid;
    enum { SERVO,
           ALIVE,
           ACK_NACK,
           NONE,
       } type;

};

#define US_LOAD_STATUS_PERIOD 141
struct us_load_status_period_msg_t {
    int msgid;
    double time;                                    // seconds
```

```
};

#define US_GENERIC_STATUS_REPORT 142
struct us_generic_status_report_t {

    int msgid;
    STATUS_TYPE  status;
    double progress;                              // percent completion
    enum {
        exception = -2,
        failed = -1,
        incomplete = 0,
        succeeded = 1,
        partial_success = 2,
        progressing = 3,
    } type;
    enum {
        exception_process_lost          = 10,
        exception_deadlock              = 11,
        exception_resource_unavailable  = 12,
        exception_resource_tip_damaged  = 13,
        exception_insufficient_capacity = 14,
    } explanation;
    void  *  command_echo;
};

//
// Errors
// First 100 are negations of Posix errno.h convention
//
#define ERRORS -200
#define US_ERROR_COMMAND_NOT_IMPLEMENTED -200
struct us_error_command_not_implemented_msg_t {
    int msgid;
};

#define US_ERROR_COMMAND_ENTRY -201
struct us_error_command_entry_msg_t {
    int msgid;
    int field_num;
};
#define US_ERROR_DUPLICATE_NAME -202
struct us_error_duplicate_name_msg_t {
    int msgid;
};

#define US_ERROR_BAD_DATA -203
struct us_error_command_bad_data_msg_t {
    int msgid;
    int field_num;
};

#define US_ERROR_NO_DATA_AVAILABLE -204
struct us_error_no_data_available_msg_t {
    int msgid;
    int field_num;
};

#define US_ERROR_SAFETY_VIOLATION  -205
struct us_error_safety_violation_msg_t {
    int msgid;
    int field_num;
};

#define US_ERROR_LIMIT_EXCEEDED  -206
```

```
struct us_error_limit_exceeded_msg_t {
    int msgid;
    Attribute_t attr;
};

#define US_ERROR_OVER_SPECIFIED  -207
struct us_error_over_specified_msg_t {
    int msgid;
    int axis_number;
};

#define US_ERROR_UNDER_SPECIFIED  -208
struct us_error_under_specified_msg_t {
    int msgid;
};

//
//
// US messages to ROBOT SERVO */
//
#define AXIS_SERVO 200

// Mode Definitions

#define US_AXIS_SERVO_USE_ANGLE_UNITS 201
struct us_axis_servo_use_angle_units_msg_t {
    int msgid;
};

#define US_AXIS_SERVO_USE_RADIAN_UNITS 202
struct us_axis_servo_use_radian_units_msg_t {
    int msgid;
};

#define US_AXIS_SERVO_USE_ABS_POSITION_MODE 203
struct  us_axis_servo_use_abs_position_mode_msg_t {
    int msgid;
};

#define US_AXIS_SERVO_USE_REL_POSITION_MODE 204
struct  us_axis_servo_use_rel_position_mode_msg_t {
    int msgid;
};

#define US_AXIS_SERVO_USE_ABS_VELOCITY_MODE 205
struct  us_axis_servo_use_abs_velocity_mode_msg_t {
    int msgid;
};

#define US_AXIS_SERVO_USE_REL_VELOCITY_MODE 206
struct  us_axis_servo_use_rel_velocity_mode_msg_t {
    int msgid;
};

#define US_AXIS_SERVO_USE_PID 207
struct  us_axis_servo_use_pid_msg_t {
    int msgid;
    int joint_mask;
};

#define US_AXIS_SERVO_USE_FEEDFORWARD_TORQUE 208
struct  us_axis_servo_use_ff_msg_t {
    int msgid;
};

#define US_AXIS_SERVO_USE_CURRENT 209
```

```
struct  us_axis_servo_use_current_msg_t {
    int msgid;
};

#define US_AXIS_SERVO_USE_VOLTAGE 210
struct  us_axis_servo_use_voltage_msg_t {
    int msgid;
};

#define US_AXIS_SERVO_USE_STIFFNESS 211
struct  us_axis_servo_use_stiffness_msg_t {
    int msgid;
    int spSelVect;                          // dof of in which to apply springs
    double * gains;                         // spring gains
    double * spMaxVel;                      // max velocity due to springs
};

#define US_AXIS_SERVO_USE_COMPLIANCE 212
struct  us_axis_servo_use_compliance_msg_t {
    int msgid;
};

#define US_AXIS_SERVO_USE_IMPEDANCE 213
struct  us_axis_servo_use_impedance_msg_t {
    int msgid;
};

#define US_AXIS_SERVO_START_GRAVITY_COMPENSATION 214
struct  us_axis_servo_start_gravity_compensation_msg_t {
    int msgid;
};

#define US_AXIS_SERVO_STOP_GRAVITY_COMPENSATION 215
struct  us_axis_servo_stop_gravity_compensation_msg_t {
    int msgid;
};

#define US_AXIS_SERVO_LOAD_DOF 216
struct  us_axis_servo_load_dof_msg_t {
    int msgid;
    int dof;
};

#define US_AXIS_SERVO_LOAD_CYCLE_TIME 217
struct  us_axis_servo_load_cycle_time_msg_t {
    int msgid;
    double time;
};

#define US_AXIS_SERVO_LOAD_PID_GAIN 218
struct us_axis_servo_load_pid_gain_msg_t {
    int msgid;
    int joint_mask;
    double  *p;                             // load proportional gain
    double  *i;                             // load integral gain
    double  *d;                             // load derivative gain
};

#define US_AXIS_SERVO_LOAD_JOINT_LIMIT 219
struct us_axis_servo_load_joint_limit_msg_t {
    int msgid;
    int axis_bit_mask;
    double  *jmaxLimit;                     // maximum joint software limits
    double  *jminLimit;                     // minimum joint software limits
};
```

```
#define US_AXIS_SERVO_LOAD_VELOCITY_LIMIT 220
struct us_axis_servo_load_velocity_limit_msg_t {
    int msgid;
    int axis_bit_mask;
    double  *jvelLimit;                          // maximum joint velocity limits
};

#define US_AXIS_SERVO_LOAD_GAIN_LIMIT 221
struct us_axis_servo_load_joint_gain_limit_msg_t {
    int msgid;
    double  *jaGain;
};

#define US_AXIS_SERVO_LOAD_DAMPING_VALUES 222
struct us_axis_servo_load_damping_values_msg_t {
    int msgid;
    double  *jaDamp;                             // damping values for impedance
};

//
// Command Data Mode
//
#define US_AXIS_SERVO_HOME 250
struct  us_axis_servo_home_msg_t {
    int msgid;
    int axis;
};

#define US_AXIS_SERVO_SET_BRAKES 251
struct  us_axis_servo_set_brakes_msg_t {
    int msgid;
    int axis_bit_mask;
};

#define US_AXIS_SERVO_CLEAR_BRAKES 252
struct  us_axis_servo_clear_brakes_msg_t {
    int msgid;
    int axis_bit_mask;
};

#define US_AXIS_SERVO_SET_TORQUE 253
struct  us_axis_servo_set_torques_msg_t {
    int msgid;
    int axis_bit_mask;
    double *joint_torques;
};

#define US_AXIS_SERVO_SET_CURRENT 254
struct  us_axis_servo_set_current_msg_t {
    int msgid;
    double *joint_currents;
};

#define US_AXIS_SERVO_SET_VOLTAGE 255
struct  us_axis_servo_set_voltage_msg_t {
    int msgid;
    double *joint_voltages;
};

#define US_AXIS_SERVO_SET_POSITION 256
struct  us_axis_servo_set_position_msg_t {
    int msgid;
    double *joint_position;
};

#define US_AXIS_SERVO_SET_VELOCITY  257
```

```
struct  us_axis_servo_set_velocity_msg_t {
    int msgid;
    double *joint_velocity;
};

#define US_AXIS_SERVO_SET_ACCELERATION  258
struct  us_axis_servo_set_acceleration_msg_t {
    int msgid;
    double *joint_acceleration;
};

#define US_AXIS_SERVO_SET_FORCES  259
struct  us_axis_servo_set_force_msg_t {
    int msgid;
    double *joint_force;
};

#define US_AXIS_SERVO_JOG 260
struct  us_axis_servo_jog_msg_t {
  int msgid;
  int axis;
  double speed;
};

#define US_AXIS_SERVO_JOG_STOP 261
struct us_axis_servo_jog_stop_msg_t {
  int msgid;
  int axis;
};


//
//
// US messages to TOOL
//
#define TOOL 300

//  Spindle
#define US_SPINDLE_RETRACT_TRAVERSE  310
struct  us_spindle_retract_traverse_msg_t {
  int msgid;
};

#define US_SPINDLE_LOAD_SPEED 311
struct  us_load_spindle_speed_msg_t {
  int msgid;
  double r;
};

#define US_SPINDLE_START_TURNING  312
struct us_start_spindle_msg_t {
  int msgid;
  enum {CLOCKWISE =  1,
        COUNTERCLOCKWISE = 2,
    } direction;
};

#define US_SPINDLE_STOP_TURNING 314
struct us_stop_spindle_turning_msg_t {
  int msgid;
};

#define US_SPINDLE_RETRACT 315
struct us_spindle_retract_msg_t {
  int msgid;
};
```

```
#define US_SPINDLE_ORIENT  316
struct us_orient_spindle_msg_t {
  int msgid;
  double orientation;
  double direction;
};

#define US_SPINDLE_LOCK_Z  317
struct us_lock_spindle_z_msg_t {
  int msgid;
};

#define US_SPINDLE_USE_FORCE 318
struct us_use_spindle_force_msg_t {
  int msgid;
};

#define US_SPINDLE_USE_NO_FORCE 319
struct us_use_no_spindle_force_msg_t {
  int msgid;
};


// Flow Control: Mist/Coolant/Abrasive Spray
#define US_FLOW_START_MIST  320
struct us_flow_start_mist_msg_t {
  int msgid;
};

#define US_FLOW_STOP_MIST 321
struct  us_flow_stop_mist_msg_t {
  int msgid;
};

#define US_FLOW_START_FLOOD 322
struct  us_flow_start_flood_msg_t {
  int msgid;
};

#define US_FLOW_STOP_FLOOD 323
struct us_flow_stop_flood_msg_t{
  int msgid;
};

#define US_FLOW_LOAD_PARAMETERS 324
struct  us_flow_load_parameters_msg_t {
  int msgid;
  enum { none,
         flow_rate,
         viscosity,
         consistency,
         thickness,
         temperature,
     } param;
    double value_rate;
    enum { beam = 0x1,
           mist=0x2,
           spray = 0x4,
       } flow;
    enum { stream,
           pulsed,
       } action;
};

//
```

```
//
// SENSOR MODULE
//
#define SENSOR 400

//
// Sensor Mode Generics
//

#define US_START_TRANSFORM   401
struct us_start_transform_msg_t {
    int msgid;
};

#define US_STOP_TRANSFORM   402
struct us_stop_transform_msg_t {
    int msgid;                                    // same as loading identity transform
};

#define US_START_FILTER  403
struct us_start_filter_msg_t {
    int msgid;
};

#define US_STOP_FILTER  404
struct us_stop_filter_msg_t {
    int msgid;                                    // same as loading no filter
};

#define US_SENSOR_USE_MEASUREMENT_UNITS 405
struct us_sensor_use_measurement_units_msg_t {
    int msgid;
    Measurement_units_type array_units;
};


//
// Sensor Parameter Generics
//

#define US_SENSOR_LOAD_SAMPLING_SPEED 406
struct us_sensor_load_sampling_speed_msg_t {
    int msgid;
    double value;
};

#define US_SENSOR_LOAD_FREQUENCY      407
struct us_sensor_load_frequency_msg_t {
    int msgid;
    double value;
};

#define US_SENSOR_LOAD_TRANSFORM   408
struct us_sensor_load_transform_msg_t {
    int msgid;
    double x,y,z,e1,e2,e3;                        // transforms
};

#define US_SENSOR_LOAD_FILTER  409
struct us_sensor_load_filter_msg_t {
    int msgid;
    enum { NONE = 0,
           LOW_PASS = 1,
           HI_PASS = 2,
       } type;
    double filter_frequency;
};

//
// Generic Commands
//
#define US_SENSOR_GET_READING 410
struct us_sensor_get_reading_msg_t {
    int msgid;
    ROUTE r;                                      // type of values: max, min, avg
                                                  // and where it goes
};

#define US_SENSOR_GET_ATTRIBUTES_READING 411
struct us_sensor_get_attributes_reading_msg_t {
    int msgid;
    ROUTE r;
    Attribute_t attr;                             // reading attributes, e.g., force | torque
};

// Not sure we need this
#define US_VECTOR_SENSOR_GET_READING 412
struct us_vector_sensor_get_reading_msg_t {
    int msgid;
    ROUTE r;

};

// Force Torque Sensor
#define US_FT_SENSOR_POST_READING 413
struct us_ft_sensor_post_reading_msg_t {
    int msgid;
    int health;
    double *f;                                    // force vector, based on dof
    double *t;                                    // torque vector, based on dof
};


// Scalar Probe
#define US_SCALAR_SENSOR_POST_READING 414
struct us_scalar_sensor_post_reading_msg_t {
    int msgid;
    double upper_limit;
    double lower_limit;
};

// 1D Vector Probe
#define US_VECTOR_SENSOR_POST_READING 415
struct us_VECTOR_sensor_post_reading_msg_t {
    int msgid;
    double *vector;
};


// Generic 2D Interface
//   e.g.,  Range or Tactile Array
//
// Mode Control to Sensor
//
#define US_2D_SENSOR_LOAD_ARRAY_PATTERN 416
struct us_2D_sensor_load_array_pattern_msg_t {
    int msgid;
    long  array_pattern;                          // bit-map of sensors enabled
    float period;                                 // period of sampling
};

#define US_2D_SENSOR_USE_ARRAY_TYPE 417
```

```
struct us_2D_sensor_use_array_type_msg_t {
    int msgid;
    enum { ONE_SHOT  = 1,
           FLOOD     = 2,
       } type;
};

//
// Input Command to 2D Sensor
//
#define US_2D_SENSOR_GET_READING 418
struct us_2D_sensor_get_reading_msg_t {
    int msgid;
    ROUTE r;
    Modifier_t mod;
};

//
// Output Data from 2D Sensor
//
#define US_2D_SENSOR_POST_READING 419
struct us_2D_sensor_post_reading_msg_t {
    int msgid;
    int rows;
    int cols;
    double  *array_values;                        // array of values
};

// Specific 2D Image Processing Sensor Interface
#define US_IMAGE_USE_FRAME_GRAB_MODE  420
struct us_image_sensor_use_frame_grab_mode_msg_t {
    int msgid;
};

#define US_IMAGE_USE_HISTOGRAM_MODE  421
struct us_image_sensor_use_histogram_mode_msg_t {
    int msgid;
};

#define US_IMAGE_USE_CENTROID_MODE  422
struct us_image_sensor_use_centroid_mode_msg_t {
    int msgid;
};

#define US_IMAGE_USE_GRAY_LEVEL_MODE  423
struct us_image_sensor_use_gray_level_mode_msg_t {
    int msgid;
};

#define US_IMAGE_USE_TRESHOLD_MODE  424
struct us_image_sensor_use_threshold_mode_msg_t {
    int msgid;
    double *threshold;
};

#define US_IMAGE_COMPUTE_SPATIAL_DERIVATIVES_MODE  425
struct us_image_sensor_compute_spatial_derivatives_msg_t {
    int msgid;
};

#define US_IMAGE_COMPUTE_TEMPORAL_DERIVATIVES_MODE  426
struct us_image_sensor_compute_temporal_derivatives_msg_t {
    int msgid;
};

#define US_IMAGE_USE_SEGMENTATATION_MODE 427
```

```
struct us_image_sensor_use_segmentation_mode_msg_t {
    int msgid;
};

#define US_IMAGE_USE_RECOGNITION_MODE  428
struct us_image_sensor_use_recognition_mode_msg_t {
    int msgid;
    OBJECT to_recognize;
};

#define US_IMAGE_COMPUTE_RANGE_MODE  429
struct us_image_sensor_compute_range_mode_msg_t {
    int msgid;
};

#define US_IMAGE_COMPUTE_FLOW_MODE  430
struct us_image_sensor_compute_flow_mode_msg_t {
    int msgid;
};

#define US_IMAGE_LOAD_CALIBRATION  431
struct us_image_sensor_calibration_msg_t {
    int msgid;
    int calibration_state;
    int cursor_value;                               // cursor value
    float cx;                                       // x center of image plane
    float cy;                                       // y center of image plane
    float sx;                                       // uncertainty scale factor
    float ncx;                                      // number of sensor elements in  camera x direction
    float nfx;                                      //  resolution of image frame - x direction
    float dx;                                       // x sensing area (designated in camera specs)/ ncx
    float dy;                                       // 2* (y sensing area)/ncy
    float dxp;                                      // dx(ncx/nfx) for camera
    float focal_length;                             // focal length of camera
    float distort;                                  //  distortion factor for camera

};

//
// Data Mode
//
#define US_IMAGE_SET_POSITION  432
struct us_image_set_sensor_position_msg_t {
    int msgid;
    float x;                                        // camera position
    float y;
    float z;
    float pan;                                      // camera orientation
    float tilt;
    float zoom;
};

#define US_IMAGE_ADJUST_POSITION  433
struct us_image_adjust_position_msg_t {
    int msgid;
                                                    // joint => 1=joint1, 2=joint2,4=joint3..
    enum { X=1,                                     // Cartesian => 1=x, 2=y, 4=z
           Y=2,                                     // depends on mode whether world or tool
           Z=4,
           PAN=5,
           TILT=6,
           ZOOM=7,
       } axis;                                      // note: no data entry
    int i;                                          // 1=increment, -1=decrement, 0=set
    double *value;                                  // if amount=0, system decides
};
```

```
#define US_IMAGE_ADJUST_FOCUS 434
struct us_image_adjust_focus_msg_t {
    int msgid;
    int i;                                          // 1=increment, -1=decrement, 0=set
    double increment;                               // if amount=0, system decides
};

#define US_IMAGE_POST_SPECIFICATION 435
struct us_image_post_specification {
    int msgid;
    STATUS_TYPE  status;
    int num_of_cameras;
    int calibration_state;
    int xpixels;
    int ypixels;
    enum {
        STATIONARY,
        MOVING,
    } type;
    TRANSFORM * base;
};

#define US_IMAGE_POST_PIXEL_MAP_READING 436
struct us_image_post_pixel_map_reading {
    int msgid;
    STATUS_TYPE status;
    TIME  timestamp;                                // reflect image data origin
    int num_cameras;                                // number of cameras
    int rows;
    int cols;
    int *image_data;                                // image data would follow here
};

#define US_IMAGE_POST_HISTOGRAM_READING 437
struct us_image_post_histogram_reading {
    int msgid;
    STATUS_TYPE status;
    TIME timestamp;                                 // reflect image data origin
    int num_cameras;                                // number of cameras
    int rows;
    int cols;
    int *image_data;                                // image data would follow here
};

#define US_IMAGE_POST_XY_CHAR_READING 438
struct us_image_post_xy_char_reading {
    int msgid;
    STATUS_TYPE status;
    TIME timestamp;                                 // reflect image data origin
    int num_cameras;                                // number of cameras
    int rows;
    int cols;
    int *image_data;                                // image data would follow here
};

#define US_IMAGE_POST_BYTE_SYMBOLIC_READING 439
struct us_image_post_byte_symbolic_reading {
    int msgid;
    STATUS_TYPE status;
    TIME timestamp;                                 // reflect image data origin
    int num_cameras;                                // number of cameras
    int rows;
    int cols;
    int *image_data;                                // image data would follow here
};
```

```
#define US_IMAGE_POST_TRESHOLD_READING 440
struct us_image_post_threshold_reading {
    int msgid;
    STATUS_TYPE status;
    TIME timestamp;                               // reflect image data origin
    int num_cameras;                              // number of cameras
    int rows;
    int cols;
    int *image_data;                              // image data would follow here
};

#define US_IMAGE_POST_SPATIAL_DERIVATIVE_READING 441
struct us_image_post_spatial_derivative_reading {
    int msgid;
    STATUS_TYPE status;
    TIME timestamp;                               // reflect image data origin
    int num_cameras;                              // number of cameras
    int rows;
    int cols;
    int *image_data;                              // image data would follow here
};

#define US_IMAGE_POST_TEMPORAL_DERIVATIVE_READING 442
struct us_image_post_temporal_derivative_reading {
    int msgid;
    STATUS_TYPE status;
    TIME timestamp;                               // reflect image data origin
    int num_cameras;                              // number of cameras
    int rows;
    int cols;
    int *image_data;                              // image data would follow here
};

#define US_IMAGE_POST_RECOGNITION_READING 443
struct us_image_post_recognition_reading {
    int msgid;
    STATUS_TYPE status;
    TIME timestamp;                               // reflect image data origin
    int num_cameras;                              // number of cameras
    int rows;
    int cols;
    int *image_data;                              // image data would follow here
};

#define US_IMAGE_POST_RANGE_READING 444
struct us_image_post_range_reading {
    int msgid;
    STATUS_TYPE status;
    TIME timestamp;                               // reflect image data origin
    int num_cameras;                              // number of cameras
    int rows;
    int cols;
    int *image_data;                              // image data would follow here
};

#define US_IMAGE_POST_FLOW_READING 445
struct us_image_post_flow_reading {
    int msgid;
    STATUS_TYPE status;
    TIME  timestamp;                              // reflect image data origin
    int num_cameras;                              // number of cameras
    int rows;
    int cols;
    int *image_data;                              // image data would follow here
};
```

```
//
//
// PIO: SENSOR, ROBOT AXIS/JOINT, TOOL Programmable Interfaces
//
#define PROGRAMMABLE_IO 500

//
// Control
//
#define US_PIO_ENABLE    500
struct us_pio_enable_msg_t {
  int msgid;
  int channel;                                    // -1 for all
};

#define US_PIO_DISABLE   501
struct us_pio_disable_msg_t {
  int msgid;
  int channel;                                    // -1 for all
};

#define US_PIO_SET_MODE  504
struct us_pio_set_mode_msg_t {
  int msgid;
  enum { INPUT=1,
         OUTPUT=2,
     } direction;
};

#define US_PIO_CONTROL_WRITE  505
struct us_pio_control_write_msg_t {
  int msgid;
                                                  // similar to control_register
                                                  // set control information
  enum { UNI_2_HALF = 1,                          // unipolar 0 to +2.5 volts
         UNI_2_HALF_NEG = 2,                      // unipolar 0 to -2.5 volts
         BI_2_HALF = 3,                           // bipolar -2.5 to 2.5 volts
         UNI_5 = 4,                               // unipolar 0 to +5 volts
         UNI_5_NEG = 5,                           // unipolar 0 to -5 volts
         BI_5 = 6,                                // bipolar -5 to 5 volts
         UNI_10 = 7,                              // unipolar 0 to +10 volts
         UNI_10_NEG = 8,                          // unipolar 0 to -10 volts
         BI_10 = 9,                               // bipolar -10 to 10 volts
         NULL_RANGE = 0,                          // Null entry
     } info;

     int bits_data;                               // 0,8,10,12,14,16,18,20,...

     enum { FREERUN = 1,
            NOFREERUN =2,
        } run;

     enum { SINGLE_END = 1,
            DIFFERENTIAL = 2,
            NULL_REF = 0,
        } ref ;
 };
#define US_PIO_LOAD_SCALE 511
struct us_pio_scale_msg_t {
  int msgid;
  int channel;
  double m ;                                      // volts to vlaue, scale factor
  double b;                                       // offset value
};
```

```
//
// Data
//
#define US_PIO_DATA_WRITE   506
struct us_pio_data_write_msg_t {
  int msgid;
  enum { SCALE, RAW } type;
  union {
        double dvalue;
        int    ivalue;
    };
};

#define US_PIO_DATA_READ    507
struct us_pio_data_read_msg_t {
  int msgid;
  enum { RAW, SCALE } type;
  int channel;
};

#define US_PIO_BIT_READ 508
struct us_pio_bit_read_msg_t {
  int msgid;
  int channel_num;
  int bit;
};

#define US_PIO_BIT_SET 509
struct us_pio_bit_set_msg_t {
  int msgid;
  int channel_num;
  int bit;
};

#define US_PIO_TOGGLE_BIT 510
struct us_pio_toggle_bit_msg_t {
  int msgid;
  int channel_num;
  int bit;
};


#define US_PIO_POST_DATA 512
struct us_pio_input_data_msg_t {
  int msgid;
  enum { RAW = 1,
         SCALED = 2,
    } type;
  union {
        unsigned long data_register;            //  data register read
        double value;
    };
    unsigned long data_mask;                    // valid bits
};

//
//
//  TLC -  TASK LEVEL CONTROL Manipulation
//

#define TASK_LEVEL_CONTROL 600

//
//
// Task Level Control
//
```

```
// Mode Selections for Reference Frames and Coordinate Chains


#define US_TLC_USE_JOINT_REFERENCE_FRAME 601
struct us_tlc_use_joint_reference_frame_msg_t {
    int msgid;
};

#define US_TLC_USE_CARTESIAN_REFERENCE_FRAME 602
struct us_tlc_use_Cartesian_reference_frame_msg_t {
    int msgid;
};

#define US_TLC_USE_REPRESENTATION_UNITS 603
struct us_tlc_use_representation_units_msg_t {
    int msgid;
    Measurement_units_type units;                 // Euler vs. Matrix Transform
};

#define US_TLC_USE_ABSOLUTE_POSITIONING_MODE 604
struct us_tlc_use_absolute_positioning_mode_msg_t {
    int msgid;                                    // aka world coordinate frame
};

#define US_TLC_USE_RELATIVE_POSITIONING_MODE 605
struct us_tlc_relative_positioning_msg_t {
    int msgid;
};

#define US_TLC_USE_WRIST_COORDINATE_FRAME 606
struct us_tlc_use_wrist_positioning_msg_t {
    int msgid;
};

#define US_TLC_USE_TOOL_TIP_COORDINATE_FRAME 607
struct us_tlc_use_tool_positioning_msg_t {
    int msgid;
};

#define US_TLC_CHANGE_TOOL 608
struct us_change_tool_msg_t {
  int msgid;
  int i;                                          // tool number
};

#define US_TLC_USE_MODIFIED_TOOL_LENGTH_OFFSETS 609
struct us_tlc_use_modified_tool_length_offsets_msg_t {
  int msgid;
  int r;
};

#define US_TLC_USE_NORMAL_TOOL_LENGTH_OFFSETS 610
struct us_tlc_use_normal_tool_length_offsets_msg_t {
  int msgid;
};

#define US_TLC_USE_NO_TOOL_LENGTH_OFFSETS 611
struct us_tlc_use_no_tool_length_offsets_msg_t {
  int msgid;
};

#define US_TLC_USE_KINEMATIC_RING_POSITIONING_MODE 612
struct us_tlc_use_kinematic_ring_msg_t {
    int msgid;
};
```

```
// Motion Modes
#define US_TLC_START_MANUAL_MOTION 613
struct us_tlc_start_manual_motion_msg_t {
    int msgid;
    AxisMask axis;
};

#define US_TLC_STOP_MANUAL_MOTION 614
struct us_tlc_stop_manual_motion_msg_t {
    int msgid;
    AxisMask axis;
};

#define US_TLC_START_AUTOMATIC_MOTION 615
struct us_tlc_start_automatic_motion_msg_t {
    int msgid;
    AxisMask axis;
};

#define US_TLC_STOP_AUTOMATIC_MOTION 616
struct us_tlc_stop_automatic__motion_msg_t {
    int msgid;
    AxisMask axis;
};

#define US_TLC_START_TRAVERSE_MOTION 617
struct us_tlc_start_traverse_motion_msg_t {
    int msgid;                                    // freespace
};

#define US_TLC_STOP_TRAVERSE_MOTION 618
struct us_tlc_stop_traverse_motion_msg_t {
    int msgid;
};

#define US_TLC_START_GUARDED_MOTION 619
struct us_tlc_start_guarded_motion_msg_t {
    int msgid;                                    // obstacle, constraints
};

#define US_TLC_STOP_GUARDED_MOTION 620
struct us_tlc_stop_guarded_motion_msg_t {
    int msgid;
};

#define US_TLC_START_COMPLIANT_MOTION 621
struct us_tlc_start_compliant_msg_t {
    int msgid;
    AxisMask axis;
    double *spring;
};

#define US_TLC_STOP_COMPLIANT_MOTION 622
struct us_tlc_stop_compliant_msg_t {
    int msgid;
    AxisMask axis;
};

#define US_TLC_START_FINE_MOTION 623
struct us_tlc_start_fine_msg_t {
    int msgid;
    AxisMask axis;
    double errtolerance;                          // amt of tolerated error in motion
    int proximity;                                // how close do we come to goal point
};
```

```
#define US_TLC_STOP_FINE_MOTION 624
struct us_tlc_stop_fine_msg_t {
    int msgid;
    AxisMask axis;
};

#define US_TLC_START_MOVE_UNTIL_MOTION 625
struct us_tlc_start_move_until_msg_t {
    int msgid;
    AxisMask axis;
    double *contact_forces;
};

#define US_TLC_STOP_MOVE_UNTIL_MOTION 626
struct us_tlc_stop_move_until_msg_t {
    int msgid;
    AxisMask axis;
};

#define US_TLC_START_STANDOFF_DISTANCE 627
struct us_tlc_start_standoff_msg_t {
    int msgid;
    AxisMask axis;
    double *distance;
};

#define US_TLC_STOP_STANDOFF_DISTANCE 628
struct us_tlc_stop_standoff_msg_t {
    int msgid;
    AxisMask axis;
};

#define US_TLC_START_FORCE_POSITIONING_MODE 629
struct us_tlc_start_force_positioning_msg_t {
    int msgid;                                    // for force reflection
};

#define US_TLC_STOP_FORCE_POSITIONING_MODE 630
struct us_tlc_stop_force_positioning_msg_t {
    int msgid;                                    // for force reflection
};

//
//
// Parameter Settings
//
#define US_TLC_LOAD_DOF 631
struct us_tlc_use_dof_msg_t {
    int msgid;
    int dof;                                      // motion DOF, i.e., 3D vs 6D
};

#define US_TLC_LOAD_CYCLE_TIME 632
struct  us_load_cycle_time_msg_t {
    int msgid;
    double time;
};

#define US_TLC_LOAD_REPRESENTATION_UNITS 633
struct us_tlc_load_representation_units_msg_t {
    int msgid;
    Measurement_units_type units;                 // Euler vs. Matrix Transform
};

#define US_TLC_LOAD_LENGTH_UNITS 634
struct us_tlc_load_length_units_msg_t {
    int msgid;
    Measurement_units_type units;                  // Meters vs. mm vs. inches
};

#define US_TLC_LOAD_RELATIVE_POSITIONING 635
struct us_tlc_load_relative_positioning_msg_t {
    int msgid;
    TRANSFORM * t;
};

#define US_TLC_ZERO_RELATIVE_POSITIONING 636
struct us_tlc_zero_relative_positioning_msg_t {
    int msgid;
};

#define US_TLC_ZERO_PROGRAM_ORIGIN 637
struct us_tlc_zerot_program_origin_msg_t {
  int msgid;
  TRANSFORM * t;
};

#define US_TLC_LOAD_KINEMATIC_RING_POSITIONING_MODE 638
struct us_tlc_load_kinematic_ring_msg_t {
    int msgid;
    Measurement_units_type units;
    enum { _Base           = 0x0000001,
           _TOOL           = 0x0000002,
           _SENSOR_FUSION  = 0x0000004,

           // RHS
           _DELTA          = 0x0000010,
           _OBJECT         = 0x0000020,

           _OBJECTBASE     = 0x0001000,
           _OBJECTOFFSET2  = 0x0002000,
           _OBJECTOFFSET3  = 0x0003000,
           _OBJECTOFFSET4  = 0x0004000,

       } mask;
};

#define US_TLC_LOAD_BASE_PARAMETERS 639
struct us_tlc_load_base_parameters_msg_t {
    int msgid;
    TRANSFORM * trBase;
};

#define US_TLC_LOAD_TOOL_PARAMETERS 640
struct us_tlc_load_tool_parameters_msg_t {
    int msgid;
    char name[128];                                // tool name
    double dx, dy, dz;                             // tooling added translation against edge
    double ux, uy, uz;                             // Euler angles for tooling angle
    double normal_threshold;                       // amount of normal force
    double tangential_threshold;                   // amount of tangential force
    ORIENTATION_TYPE heading;                      // what is the heading of the tool tip
};

#define US_TLC_LOAD_OBJECT 641
struct us_tlc_load_object_msg_t {
    int msgid;
    OBJECT obj_id;
    TRANSFORM * t;
};

#define US_TLC_LOAD_OBJECT_BASE 642
```

```
struct us_tlc_load_object_base_msg_t {
    int msgid;
    TRANSFORM * t;

};

#define US_TLC_LOAD_OBJECT_OFFSET 643
struct us_tlc_load_object_offset_msg_t {
    int msgid;
    int i;
    TRANSFORM * t;
};

#define US_TLC_LOAD_DELTA 644
struct us_tlc_load_delta_msg_t {
    int msgid;
    enum { _SINE_WAVE_,
           _DITHER_,
           _NULL_,
    } delta;
    double magnitude;
    double frequency;
};

#define US_TLC_LOAD_OBSTACLE_VOLUME 645
struct us_tlc_load_obstacle_volume_msg_t {
    int msgid;
    int i;
    TRANSFORM * t;
};

// Dynamical Control
#define US_TLC_LOAD_NEIGHBORHOOD 646
struct us_tlc_load_blending_msg_t {
    int msgid;
    double dist;                                 // error tolerance
    BLEND_TYPE blend;                            // what is the blending algorithm
};

#define US_TLC_LOAD_FEED_RATE 647
struct us_tlc_load_feed_rate_msg_t {
    int msgid;
    double feed_rate;
    Measurement_units_type units;
};

#define US_TLC_LOAD_TRAVERSE_RATE 648
struct us_tlc_load_traverse_rate_msg_t {
    int msgid;
    double traverse_rate;
    Measurement_units_type units;
};

#define US_TLC_LOAD_ACCELERATION 649
struct us_tlc_load_acceleration_msg_t {
    int msgid;
    double accel;
    Measurement_units_type units;
};

#define US_TLC_LOAD_JERK 650
struct us_tlc_load_jerk_msg_t {
    int msgid;
    double jerk;
    Measurement_units_type units;
};
```

```
#define US_TLC_LOAD_PROXIMITY 651
struct us_tlc_load_proximity_msg_t {
    int msgid;
    AxisMask axis;
    double distance;
};

#define US_TLC_LOAD_CONTACT_FORCES 652
struct us_tlc_load_contact_forces_msg_t {
    int msgid;
    TRANSFORM * tr;                            // transform from MERGE frame
                                               // to FORCE frame
    Representation_units_type units;           // transform rep.
    int dof;                                   // degrees of freedom
    long cfSelVect;                            // hybrid selection vector for
                                               // FORCE frame
    long cfComplyVect;                         // selection vector specifying
                                               // which position DOFs of FORCE
                                               // frame also have compliance
    double *cfFtSetpoints;                     // force setpoints in force
                                               // controlled DOFs of FORCE frame
    double *cfFtGains;                         // force gains in FORCE frame
    double *cfMaxFcVel;                        // max velocities in DOF of
                                               // force frame due to force control
};

#define US_TLC_LOAD_JOINT_LIMIT 653
struct us_tlc_load_joint_limit_msg_t {
    int msgid;
    AxisMask axis;
    double  *jtLimit;                          // joint space limit
};

#define US_TLC_LOAD_CONTACT_FORCE_LIMIT 654
struct us_tlc_load_contact_force_limit_msg_t {
    int msgid;

    double  *ctFLimit;                         // contact force limit
};

#define US_TLC_LOAD_CONTACT_TORQUE_LIMIT 655
struct us_tlc_load_contact_torque_limit_msg_t {
    int msgid;
    double  *ctTLimit;                         // contact torque limit
};

#define US_TLC_LOAD_SENSOR_FUSION_POS_LIMIT 656
struct us_tlc_load_sensor_fusion_pos_limit_msg_t {
    int msgid;
    double *fsPLimit;                          // position limit for sensor based motion
};

#define US_TLC_LOAD_SENSOR_FUSION_ORIENT_LIMIT 657
struct us_tlc_load_sensor_fusion_orient_limit_msg_t {
    int msgid;
    double *fsOLimit;                          // orientation limit for sensor based motion
};

#define US_TLC_LOAD_SEGMENT_TIME 658
struct us_tlc_load_segment_time_msg_t {
    int msgid;
    double time;                               // duration of segment
};

#define US_TLC_LOAD_TERMINATION_CONDITION 659
```

```
struct us_tlc_load_termination_condition_msg_t {
    int msgid;
    enum { time_term    =     0x01,
           time_max     =     0x02,
           trans_del    =     0x04,
           ang_del      =     0x08,
           force_err    =     0x10,
           torque_err   =     0x20,
           vel_profile  =     0x40,
       } condition;

    int select;                                  // bit mask for termination condition
    double testTime;                             // time over which to avg ending condition
                                                 // variables
    double endTime;                              // maximum ending motion time
    double endTransDel;                          // total translation due to sensor based
                                                 // motion in MERGE frame
    double endAngDel;                            // total angular motion due sensor based motion
                                                 // motion in MERGE frame
    double endTransVel;                          // magnitude of rate of change of endTransDel
    double endAngVel;                            // magnitude of rate of change of endAngDel
    double endForceErr;                          // contact force error vector magnitude
    double endTorqueEff;                         // contact torque error vector magnitude
    double endForceVel;                          // magnitude of raet of change of endForceErr
    double endTorqueVel;                         // magnitude of raet of change of endTorqueErr
};

#define US_TLC_INCR_VELOCITY 660
struct us_tlc_incr_velocity_msg_t {
    int msgid;
    int i;                                       // 1=increment, -1=decrement, 0=set
    double increment;                            // if amount=0, system decides
};

#define US_TLC_INCR_ACCELERATION 661
struct us_tlc_incr_acceleration_msg_t {
    int msgid;
    int i;                                       // 1=increment, -1=decrement, 0=set
    double increment;                            // if amount=0, system decides
};


//
// Task Level Control
//
// Command Data
//
#define US_TLC_SET_GOAL_POSITION 662
struct us_tlc_set_goal_position_msg_t {
    int msgid;
    double *data;
};

#define US_TLC_GOAL_SEGMENT 663
struct us_tlc_goal_segment_msg_t {
    int msgid;
    SEGMENT_SELECT *segment;                     // segment type & description
};

#define US_TLC_ADJUST_AXIS 664
struct us_tlc_adjust_axis_msg_t {
    int msgid;
    AxisMask axis;
    int i;                                       // 1=increment, -1=decrement, 0=set
    double *value;                               // if amount=0, system decides
};
```

```
// Status Data
#define US_TLC_UPDATE_SENSOR_FUSION 665
struct us_tlc_update_sensor_fusion_msg_t {
    int msgid;
    TRANSFORM * update;
};


//
//
// TLC:: task level control : cutting/machining
//

#define US_TLC_SELECT_PLANE 666
struct  us_tlc_select_plane_msg_t {
    int msgid;
    AxisMask  axis;
};

#define US_TLC_USE_CUTTER_RADIUS_COMPENSATION 667
struct us_tlc_use_cutter_radius_compensation_msg_t {
    int msgid;
    double radius;
};

#define US_TLC_START_CUTTER_RADIUS_COMPENSATION 668
struct us_tlc_start_cutter_radius_compensation_msg_t {
    int msgid;
    double side;
};

#define US_TLC_STOP_CUTTER_RADIUS_COMPENSATION 669
struct us_tlc_stop_cutter_radius_compensation_msg_t {
    int msgid;
};

#define US_TLC_STRAIGHT_TRAVERSE 670
struct us_tlc_straight_traverse_msg_t {
    int msgid;
    double x;
    double y;
    double z;
};

#define US_TLC_ARC_FEED 671
struct us_tlc_arc_feed_msg_t {
    int msgid;
    AxisMask first_axis;
    AxisMask second_axis;
    double rotation;
    double axis_end_point;
};

#define US_TLC_STRAIGHT_FEED  672
struct us_tlc_straight_feed_msg_t {
    int msgid;
    double x;
    double y;
    double z;
};

#define US_TLC_PARAMETRIC_2D_CURVE_FEED 673
struct us_tlc_parametric_2d_curve_feed_msg_t {
    int msgid;
    FUNCTION_PTR f1;
    FUNCTION_PTR f2;
    double start_parameter_value;
    double end_parameter_value;
};

#define US_TLC_PARAMETRIC_3D_CURVE_FEED 674
struct us_tlc_parametric_3d_curve_feed_msg_t {
    int msgid;
    FUNCTION_PTR xfcn;
    FUNCTION_PTR yfcn;
    FUNCTION_PTR zfcn;
    double start_parameter_value;
  double end_parameter_value;
};

#define US_TLC_NURBS_KNOT_VECTOR 675
struct us_tlc_nurbs_knot_vector_msg_t {
    int msgid;
    int i;                                      // which element, 0 = first
    double k;
};

#define US_TLC_NURBS_CONTROL_POINT  676
struct us_tlc_nurbs_control_point_msg_t {
    int msgid;
    int i;                                      // which CP, 0 = first
    double x;
    double y;
    double z;
    double w;                                   // the weight
};

#define US_TLC_NURBS_FEED 677
struct  us_tlc_nurbs_feed_msg_t {
    int msgid;
    double sStart;
    double sEnd;
};

#define US_TLC_TELEOP_FORCE_REFLECTION_UPDATE  678
struct us_tlc_teleop_force_reflection_msg_t{
    int msgid;
    double *data;
};



//




//////////////////////////////////////////////////////////////////////////
//
//
//                              L O C A L
//
//
//////////////////////////////////////////////////////////////////////////



//                  **     DISCLAIMER II     **
//
```

```
// The following LOCAL interfaces are preliminary. The LOCAL interfaces
// are an initial attempt at providing a solution. These interfaces have
// not undergone the necessary peer-review process. Please do not let the
// preliminary state of these interfaces reflect too negatively on the
// overall state of the UTAP interfaces.  At some point in the future,
// these interfaces will undergo the scrutiny of a review panel and will
// receive the same level of discussion and revision that was given to
// the LOCAL interfaces.  Comments concerning the LOCAL interfaces are
// welcome, and should be directed to the UTAP interface coordinator,
// listed on the disclaimer page.

//
// At some point the feature-based concepts of the APT Part Programming
// Language will be explicitly incorporated into the LOCAL interfaces,
// specifically the APT Tool Axis Control Language, and the APT Measure
// Language. APT contains hooks for Robotics and Vision Commands (Rules 14xx).
//
// The current emphasis of the LOCAL definitions is to establish the
// framework in which the operator can make selections and have these
// selections registered in the control system.
//


//
// TDS - the task description module commands/controls task
#define TASK_DESCRIPTION 1000

#define US_TDS_LOAD_USER  1000
struct us_tds_load_user_msg_t{
    int msgid;
    USER_TYPE user;                               // limit programming capabilities
};

#define US_TDS_SELECT_PROGRAM  1001
struct us_tds_select_program_msg_t{
    int msgid;
    char filename[128];                           // filename on disk
};

#define US_TDS_EXECUTE_PROGRAM  1002
struct us_tds_execute_program_msg_t{
    int msgid;
    char filename[128];                           // filename on disk
};

#define US_TDS_SELECT_OPERATION  1003
struct us_tds_select_operation_msg_t{
    int msgid;
    enum {  _move              = 1,
            _paint             = 2,
            _strip             = 3,
            _finish            = 4,
            _polish            = 5,
            _clean             = 6,
            _deseal            = 7,
            _seal              = 8,
            _inspect           = 9,
            _cut               = 10,
        } task;
};

#define US_TDS_SELECT_OPMODE  1004
struct us_tds_select_opmode_msg_t{
    int msgid;
    enum {
        TELEOP,                                   // joystick motion
};
        SUPERVISED,                         // operator supervises actions
        AUTONOMOUS,                         // controller makes crucial decision
        TRADED,                             // traded control of motion
        SHARED                              // control of axis of motion is shared
    } type;                                 // type of operator interaction
    AxisMask axis;
};

#define US_TDS_LOAD_SELECTIONS 1005
struct us_tds_load_selections_msg_t{
    int msgid;
    enum { select_agent,
           select_io,
           select_object,
           select_traj,
       }selection;
    char name[128];
};

#define US_TDS_LOAD_REFERENCE_UNITS  1006
struct us_tds_load_reference_units_msg_t{
    int msgid;
    Measurement_units_type units;
};

#define US_TDS_LOAD_RATE_DEFAULTS  1007
struct us_tds_load_rates_msg_t{
    int msgid;
    enum {
        set_default_feed_rate,              // per second
        set_default_traverse_rate,          // per second
        set_task_space_acceleration_limit   // per second per second
    } selection;

    enum { meters,
           inches,
           millimeters } units;

    double rate;
};

#define US_TDS_LOAD_ORIGIN  1008
struct us_tds_load_origin_msg_t{
    int msgid;
    enum { device_origin,                   // use current values of device
           relative_origin,
           zero_device,
           device_view,
       } selection;
    char name [128];                        // device name
};

#define US_TDS_LOAD_SENSING_DEFAULTS  1009
struct us_tds_load_sensing_msg_t{
    int msgid;
    enum {
        set_default_sensor_limit,
        set_default_sensor_orientation,
        set_sensor_limit_override,
        clear_sensor_override
    } selection;
    char  sensor_name[128];
    Attribute_t attr;
    double setting;
};
```

```
//
//
// TK - The current state of the manipulation, end-effecting, and tooling
//   systems is known and stored in the task knowledgebase and trajectory
//   description modules

#define TASK_KNOWLEDGE 1100

#define US_TK_DEFINE_FRAMEWORK 1101
struct us_task_framework_msg_t {
    int msgid;

    // -1 indicates that the user must fill in the field
    enum {  _move,
            _paint,
            _strip,
            _finish,
            _polish,
            _clean,
            _deseal,
            _seal,
            _inspect,
        } task;

    int step_number;                        // use step number or
    char macro_name[128];                   // task macro name


    USER_TYPE user;                         // minimum programming capabilities

    // Select Operation Method
    enum { TELEOP,                          // joystick motion
           SUPERVISED,                      // operator supervises actions
           AUTONOMOUS,                      // controller makes crucial decision
           TRADED,                          // traded control of motion
           SHARED,                          // control of axis of motion is shared
       } type;                              // type of operator interaction

    AxisMask axis;

    int number_of_agents;                   // number of agents agents defaults
    char  agent_class[128][100];
    char  agent_list[128][100];

    char tool_class[128];                   // class of potential tools
    char tool_name[128];                    // default tool

    char object_class[128];                 // attribute class  of potential objects
    char object_name[128];                  // use selects/defines object
    int task_units;                         // default units

    POSITION program_home;
    POSITION program_origin;

    POSITION relative_origin;

    TRANSFORM   * base_frame;
    TRANSFORM   * tool_frame;

    TRANSFORM   * zero_axes_force;
    TRANSFORM   * zero_tool_force;

    int default_task_reference_units;
    int task_reference_units;
    double set_task_space_acceleration_limit;
    double set_task_space_acceleration_time;

    double feed_rate;
    double feed_rate_units;

    double traverse_rate;
    int traverse_rate_units;

    double default_force_setting;
    double guarded_proximity_setting;
    double viscosity_setting;
    double humidity_setting;
    double desired_temperature;
    double temperature_limit;
    double noise_limit;


};


#define US_TK_MACRO_CREATE 1102
#define US_TK_MACRO_DELETE 1103
#define US_TK_MACRO_MODIFY 1104
struct  us_tk_macro_msg_t {
  int msgid;
  char framework_file [128];                    // defines framework
  char action_file [128];                       // defines stepwise actions
  char plan[128];
};


//
// PTPS/TPS
#define PARENT_TASK_PROGRAM_SEQUENCING 1300

#define US_PTPS_SELECT_AGENT  1301
struct us_select_resource_msg_t {
    int msgid;
    TASK_ID tid;
    RESOURCE_SELECT agent;
    SUBSYSTEM_ID ssid;
    enum { SOLO,
           LH,
           RH,
       } type;
};

#define US_TPS_SELECT_TOOL 1302
struct us_select_tool_msg_t {
    int msgid;
    TASK_ID tid;
    END_EFFECTOR_SELECT tool;
    SUBSYSTEM_ID ssid;
};

#define US_PTPS_SELECT_SENSOR 1303
struct us_select_sensor_msg_t {
    int msgid;
    TASK_ID tid;
    RESOURCE_SELECT agent;
    SUBSYSTEM_ID ssid;
    enum { SOLO,
           LH,
           RH,
       } type;
```

```
};

#define US_PTPS_INTERP_RUN_PLAN 1303
struct  us_interp_run_plan_msg_t {
  int msgid;
  SUBUSYSTEM_ID ssid;
  enum { UTAP      = 1,
         RS274D    = 2,
         SIL       = 3,
         GSL       = 4,
     } type ;
  char plan[128];
};

#define US_PTPS_INTERP_HALT_PLAN 1304
struct us_interp_halt_plan_msg_t {
    int msgid;
    SUBUSYSTEM_ID ssid;

};

#define US_PTPS_INPUT_REQUEST 1305
struct us_ptps_input_request_msg_t {
    int msgid;
    SUBUSYSTEM_ID ssblocker;
    SUBUSYSTEM_ID ssenabler;
    enum {  peer_ack          = 1,
            peer_done         = 2,
            shared_resource   = 3,
        } type;
};

#define US_PTPS_OUTPUT_ENABLE_SUBSYSTEM 1306
struct us_ptps_output_enable_msg_t {
    int msgid;
    SUBUSYSTEM_ID ssblocker;
    SUBUSYSTEM_ID ssenabler;
    enum {  peer_ack          = 1,
            peer_done         = 2,
            shared_resource   = 3,
        } type;
};

//
// TPS
#define TASK_PROGRAM_SEQUENCING 1400


#define US_TPS_FREESPACE_MOTION 1401
struct us_tps_freespace_msg_t {
    int msgid;
};

#define US_TPS_GUARDED_MOTION 1402
struct us_tps_guardede_msg_t {
    int msgid;
};

#define US_TPS_CONTACT_MOTION 1403
struct us_tps_constact_msg_t {
    int msgid;
};

#define US_TPS_SET_SUPERVISORY_MODE 1404
struct us_supervisory_mode_msg_t {
    int msgid;
    // need hybrid parameter stuff here
};


#define US_TPS_SELECT_FEATURE 1405
struct us_select_feature_msg_t {
    int msgid;
    FEATURE surface;
    double fx,fy,fz;                        // world to feature origin translation
    double fo1,fo2,fo3;                     // world to feature origin rotation
};

#define US_TPS_SELECT_MATERIAL 1406
struct us_select_material_msg_t {
    int msgid;
    MATERIAL_TYPE m;                        // type of material
    double maxx,maxy,maxz;                  // feature to operation max translation
    double minx,miny,minz;                  // feature to operation min translation
    double fo1,fo2,fo3;                     // feature to operation max rotation
    double strength;                        // maximum material strength
    double minforce;                        // min amount of surface contact?
    double maxforce;                        // max amount of surface contact?
};


#define US_LOAD_OBSTACLE 1407
struct us_load_obstacle_msg_t {
    int msgid;
    FEATURE obstacle;
};

#define US_LOAD_PATTERN 1408
struct us_load_pattern_msg_t {
    int msgid;
    GEOMETRY_PATTERN pattern;
};


#define US_TPS_MARK_EVENT 1409
struct us_tps_mark_event_msg_t {
    int msgid;
    enum {
        peer_signal = 1,
        // coordinate devices/io/sensed motion
    } event;
};

#define US_TPS_ENABLE 1410
struct us_ptps_enable_msg_t {
    int msgid;
    enum {  peer_msg          = 1,
            shared_resource   = 2,
        }enable;
};


//
// OI  - Operator Interface Messages
#define OPERATOR_INTERFACE 1500

#define US_BEGIN_FRAMEWORK 1501
#define US_END_FRAMEWORK  1502
#define US_CREATE_FRAMEWORK  1503
#define US_DELETE_FRAMEWORK  1504
struct us_framework_msg_t{
    int msgid;
    char name [128];
};

#define US_ADD_SYMBOLIC_ITEM 1505
#define US_DELETE_SYMBOLIC_ITEM  1506
struct us_symbolic_item_msg_t{
    int msgid;
    char name [128];
};

#define US_ADD_SYMBOLIC_ITEM_ATTR 1507
#define US_DELETE_SYMBOLIC_ITEM_ATTR  1508
#define US_SET_SYMBOLIC_ITEM_ATTR 1509
struct us_symbolic_item_attribute_msg_t{
    int msgid;
    char name [128];
    char attribute_name[128];
    int size;                                     // e.g. number of joints
    int xdim;
    int ydim;
    Representation_units_type  rep;
    Measurement_units_type   units;
    generic_value_a values;                       // context-dependent values

};

//
// OM  - object modeling module
#define OBJECT_MODELING 1600

#define US_OM_CREATE  1601
struct us_om_create_msg_t {
    int msgid;
    enum { device_origin = 1,
           relative_origin = 2,
           zero_device = 3,
           device_view = 4,
           workarea = 5,
           target   = 6,
           obstacle = 7,
       } type;

    char name [128];
    // Reference Frame - e.g., given in VDT relative coordinates
    char device[128];                             // use name for now
    GEOMETRY data;                                // define shape
};

#define US_OM_DELETE  1602
struct us_om_delete_msg_t {
    int msgid;
    enum { device_origin = 1,
           relative_origin = 2,
           zero_device = 3,
           device_view = 4,
           workarea = 5,
           target   = 6,
           obstacle = 7,
       } type;

    char name [128];
};

#define US_OM_MODIFY  1603
struct us_om_modify_msg_t {
    int msgid;
    enum { device_origin = 1,
           relative_origin = 2,
           zero_device = 3,
           device_view = 4,
           workarea = 5,
           target   = 6,
           obstacle = 7,
       }type;

    char name [128];
    // Reference Frame - e.g., given in VDT relative coordinates
    char device[128];                              // use name for now
    GEOMETRY data;                                 // define shape

};

//
// OC - The object calibration module provides the operator with a means
// of updating knowledge on the object(s) positions and orientations
#define OBJECT_CALIBRATION 1700

#define US_OC_SET_CALIB  1701
#define US_OC_GET_CALIB 1702
struct us_oc_calib_msg_t {
    int msgid;
    enum { device_origin = 1,
           relative_origin = 2,
           zero_device = 3,
           device_view = 4,
           workarea = 5,
           target   = 6,
           obstacle = 7,
       }type;

    char name [128];
    // Reference Frame - e.g., given in VDT relative coordinates
    char device[128];                              // use name for now
    GEOMETRY data;                                 // define shape
};

#define US_OC_SET_ATTR  1703
struct us_oc_set_attr_msg_t {
    int msgid;
    char name [128];                               // device name
    Modifier_t modifier;
    Attribute_t attributes;
    int size;
    Representation_units_type  rep;
    Measurement_units_type units;
    generic_value_a value;
};

#define US_OC_GET_ATTR  1704
struct us_oc_get_attr_msg_t {
    int msgid;
    char name [128];                               // device name
    Modifier_t modifier;
    Attribute_t attributes;
};

//
// OK Input
#define OBJECT_KNOWLEDGE 1800

#define US_OK_RECORD   1801
#define US_OK_PLAYBACK 1802
```

```
struct us_ok_record_msg_t {
    int msgid;
    char name [128];
};

#define US_OK_CREATE_OBJ 1803
struct us_ok_create_msg_t {
    int msgid;
    char name [128];
    OBJECT ob;
};

#define US_OK_DELETE_OBJ 1804
struct us_ok_delete_msg_t {
    int msgid;
    char name [128];
};

#define US_OK_MODIFY 1805
struct us_ok_modify_msg_t {
    int msgid;
    int obj_id;
    int size;
    void * data;
};

#define US_OK_MODIFY_ATTRIBUTE 1806
struct us_ok_modify_attribute_msg_t {
    int msgid;
    int obj_id;
    Attribute_t attr;
    int size;
    void * data;
};

#define US_OK_ATTRIBUTE_QUERY 1807
struct us_ok_attr_query_msg_t {
    int msgid;
    int obj_id;
    Attribute_t attr;
};

// Output
#define US_OK_OUTPUT_REGISTERED_OBJ_ID  1808
struct us_registered_id_msg_t {
    int msgid;
    char name [128];
    int obj_id;
};

#define US_OK_ATTRIBUTE_RESPONSE 1809
struct us_ok_attr_response_msg_t {
    int msgid;
    int obj_id;
    Attribute_t attr;
    double  *values;
};

//
// TRD - the trajectory description module suports the creation,
// deletion or modification of a trajectory
#define TRAJECTORY_DESCRIPTION 1200

#define US_TRD_OPEN 1200
struct us_trd_open_msg_t{
    int msgid;
    char name[128];
    enum { create   = 1,
           append   = 2,
           readonly = 3,
       } type;
};

#define US_TRD_ERASE 1201
#define US_TRD_RECORD 1202
#define US_TRD_RECORD_ON 1203
#define US_TRD_RECORD_OFF 1204
struct us_trd_record_msg_t{
    int msgid;
    char name[128];
};



#define US_TRD_FIND 1205
#define US_TRD_NEXT 1206
#define US_TRD_PREVIOUS 1207
#define US_TRD_DELETE  1208
struct us_trd_positioning_msg_t{
    int msgid;
    char name[128];
    int num_element;                              // -1 = current
};

#define US_TRD_NAME_ITEM 1209
struct us_trd_name_item_msg_t{
    int msgid;
    char name[128];
};

#define US_TRD_DELETE_ITEM  1210
struct us_trd_delete_item_msg_t{
    int msgid;
    int id;
};

#define US_TRD_SET_JOINT_MODE 1211
struct us_trd_set_joint_mode_msg_t{
    int msgid;
    double dof;
};

#define US_TRD_SET_CARTESIAN_MODE 1212
struct us_trd_set_Cartesian_mode_msg_t{
    int msgid;
    double dof;
};

#define US_TRD_MODIFY  1213
struct us_trd_modify_msg_t{
    int msgid;
    char name[128];
    int num_element;
    double *data;
};

#define US_TRD_ADD_ELEMENT  1214
struct us_trd_add_element_msg_t{
    int msgid;
    double *data;
};
```

```
//
// SGD | ADS - Analysis and Device Simulator Modules. These modules serve
// a dual purpose: 1) operator can call the analysis menu, etner state
// data and end point data, and let the simulator establish the
// appropriate trajectory/path through teleoperation of the simulation
// 2) analyzes the exeuction of the system taks sequence by examining
// the curernt state of teh system against predetermined constraints.
// SS - subsystem simulator

// SGD uses the same messaging as the OI

// ADS uses the same messaging as the SGD, OI
#define STATUS_GRAPHICS_DISPLAY 1900
#define ANALYSIS_DIAGNOSIS_SYSTEM 2000

#define US_ADS_COLLISION_DETECTED 2001
struct us_sgd_error_msg_t {
    int msgid;
    char name [128];
    int obj_id1;
    int obj_id2;
    double x,y,z;                                // collision_spot
};

//
//
// SS uses the same messaging as the module it is simulating but replace
// a SS for the module name.
#define SUBSYSTEM_SIMULATION 2100



#endif
```

H.4 Interface API Source

```
#ifndef UTAP_INTERFACE_DEFINITIONS
#include "generic_defs.h"
#include "utap_info_model.h"
#include "utap_data_defs.h"
#include "undefined_types.h"
/**
   This header file defines the interfaces for communication between
   modules in the Generic C5 Architecture.
   Generic Telerobotic Architecture for C-5 Industrial Processes
   contains  modules of which the following have acronyms:
   MODULES NAMING/ACRONYMGS:
   REMOTE:
     RSC   - robot servo control
     TOOL  - tool control
     SENSOR- sensor control
     PIO   - programmable io
     TLC   - task level control
     CLC   - closed loop control
     DB    - data base is part of TLC & CLC
     VS    - virtual sensor
  LOCAL:
     TDS   - task description and supervision
     TK    - task knowledge
     TRD   - trajectory description
     PTPS  - parent task program  sequencer
     TPS   - task program  sequencer
     OI    - operator interface
     OK    - object knowledge
     OC    - object calibration
     OM    - object modeling
     SGD   - status graphics displays
     SS    - subsystem simulators
     ADS   - analysis and diagnosis
  */
/**
 A little table of contents:
  GENERIC:        101  -   199
  ERROR:         -100  -  -200
  ROBOT_SERVO:    200  -   299
  TOOL:           300  -   399
  SENSOR:         400  -   499
  PIO:            500  -   599
  TLC:            600  -   699
  DB:             700  -   799
  VS:             800  -   899
  TDS:           1000  -  1099
  TK:            1100  -  1199
  TRD:           1200  -  1299
  PTPS:          1300  -  1399
  TPS:           1400  -  1499
  OI:            1500  -  1599
  OK             1600  -  1699
  OC             1700  -  1799
  OM             1800  -  1899
  SGD:           1900  -  1999
  ADS:           2000  -  2099
  SS:            2100  -  2199
  */
us_startup();
us_shutdown();
us_reset( int  type,
          long  mask);
us_enable( int axis);
us_disable( int axis);
us_estop();
us_start();
us_stop();
us_abort();
us_halt();
us_init();
us_hold();
us_pause();
us_resume();
us_zero( long mask);
us_begin_single_step();
us_next_single_step();
us_clear_single_step();
us_begin_block();
us_end_block();
us_begin_plan( char name[128]);
us_end_plan();
us_use_plan();
us_begin_macro( char name[128]);
us_end_macro();
us_execute_macro( char name[128]);
us_begin_event( char name[128],
                int  type,
                TIME t);
us_end_event();
us_mark_breakpoint();
us_mark_event( char name[128]);
us_get_selection_id( char name[128]);
us_post_selection_id();
```

```
us_use_selection();
us_use_axis_mask( AxisMask axis);
us_use_ext_algorithm( int slot);
us_load_ext_parameter( int slot);
us_get_ext_data_value( int slot,
                       ROUTE r);
us_post_ext_data_value( int slot,
                        void * data);
us_set_ext_data_value( int slot,
                       void * data);
us_load_status( int  type);
us_load_status_period( double time);
us_generic_status_report_t( STATUS_TYPE  status,
                            double progress,
                            int  type,
                            int  explanation,
                            void  *  command_echo);
us_error_command_not_implemented();
us_error_command_entry( int field_num);
us_error_duplicate_name();
us_error_command_bad_data( int field_num);
us_error_no_data_available( int field_num);
us_error_safety_violation( int field_num);
us_error_limit_exceeded( Attribute_t attr);
us_error_over_specified( int axis_number);
us_error_under_specified();
us_axis_servo_use_angle_units();
us_axis_servo_use_radian_units();
us_axis_servo_use_abs_position_mode();
us_axis_servo_use_rel_position_mode();
us_axis_servo_use_abs_velocity_mode();
us_axis_servo_use_rel_velocity_mode();
us_axis_servo_use_pid( int joint_mask);
us_axis_servo_use_ff();
us_axis_servo_use_current();
us_axis_servo_use_voltage();
us_axis_servo_use_stiffness( int spSelVect,
                             double * gains,
                             double * spMaxVel);
us_axis_servo_use_compliance();
us_axis_servo_use_impedance();
us_axis_servo_start_gravity_compensation();
us_axis_servo_stop_gravity_compensation();
us_axis_servo_load_dof( int dof);
us_axis_servo_load_cycle_time( double time);
us_axis_servo_load_pid_gain( int joint_mask,
                             double  *p,
                             double  *i,
                             double  *d);
us_axis_servo_load_joint_limit( int axis_bit_mask,
                                double  *jmaxLimit,
                                double  *jminLimit);
us_axis_servo_load_velocity_limit( int axis_bit_mask,
                                   double  *jvelLimit);
us_axis_servo_load_joint_gain_limit( double  *jaGain);
us_axis_servo_load_damping_values( double  *jaDamp);
us_axis_servo_home( int axis);
us_axis_servo_set_brakes( int axis_bit_mask);
us_axis_servo_clear_brakes( int axis_bit_mask);
us_axis_servo_set_torques( int axis_bit_mask,
                           double *joint_torques);
us_axis_servo_set_current( double *joint_currents);
us_axis_servo_set_voltage( double *joint_voltages);
us_axis_servo_set_position( double *joint_position);
us_axis_servo_set_velocity( double *joint_velocity);
us_axis_servo_set_acceleration( double *joint_acceleration);
```

```
us_axis_servo_set_force( double *joint_force);
us_axis_servo_jog( int axis,
                   double speed);
us_axis_servo_jog_stop( int axis);
us_spindle_retract_traverse();
us_load_spindle_speed( double r);
us_start_spindle( int  direction);
us_stop_spindle_turning();
us_spindle_retract();
us_orient_spindle( double orientation,
                   double direction);
us_lock_spindle_z();
us_use_spindle_force();
us_use_no_spindle_force();
us_flow_start_mist();
us_flow_stop_mist();
us_flow_start_flood();
us_flow_stop_flood();
us_flow_load_parameters( int  param,
                         double value_rate,
                         int  flow,
                         int  action);
us_start_transform();
us_stop_transform();
us_start_filter();
us_stop_filter();
us_sensor_use_measurement_units( Measurement_units_type array_units);
us_sensor_load_sampling_speed( double value);
us_sensor_load_frequency( double value);
us_sensor_load_transform( double x,
                          y,
                          z,
                          e1,
                          e2,
                          e3);
us_sensor_load_filter( int  type,
                       double filter_frequency);
us_sensor_get_reading( ROUTE r);
us_sensor_get_attributes_reading( ROUTE r,
                                  Attribute_t attr);
us_vector_sensor_get_reading( ROUTE r);
us_ft_sensor_post_reading( int health,
                           double *f,
                           double *t);
us_scalar_sensor_post_reading( double upper_limit,
                               double lower_limit);
us_VECTOR_sensor_post_reading( double *vector);
us_2D_sensor_load_array_pattern( long  array_pattern,
                                 float period);
us_2D_sensor_use_array_type( int  type);
us_2D_sensor_get_reading( ROUTE r,
                          Modifier_t mod);
us_2D_sensor_post_reading( int rows,
                           int cols,
                           double  *array_values);
us_image_sensor_use_frame_grab_mode();
us_image_sensor_use_histogram_mode();
us_image_sensor_use_centroid_mode();
us_image_sensor_use_gray_level_mode();
us_image_sensor_use_threshold_mode( double *threshold);
us_image_sensor_compute_spatial_derivatives();
us_image_sensor_compute_temporal_derivatives();
us_image_sensor_use_segmentation_mode();
us_image_sensor_use_recognition_mode( OBJECT to_recognize);
us_image_sensor_compute_range_mode();
us_image_sensor_compute_flow_mode();
```

```
us_image_sensor_calibration( int calibration_state,
                             int cursor_value,
                             float cx,
                             float cy,
                             float sx,
                             float ncx,
                             float nfx,
                             float dx,
                             float dy,
                             float dxp,
                             float focal_length,
                             float distort);
us_image_set_sensor_position( float x,
                              float y,
                              float z,
                              float pan,
                              float tilt,
                              float zoom);
us_image_adjust_position( int  axis,
                          int i,
                          double *value);
us_image_adjust_focus( int i,
                       double increment);
us_image_post_specification( STATUS_TYPE  status,
                             int num_of_cameras,
                             int calibration_state,
                             int xpixels,
                             int ypixels,
                             int  type,
                             TRANSFORM * base);
us_image_post_pixel_map_reading( STATUS_TYPE status,
                                 TIME  timestamp,
                                 int num_cameras,
                                 int rows,
                                 int cols,
                                 int *image_data);
us_image_post_histogram_reading( STATUS_TYPE status,
                                 TIME timestamp,
                                 int num_cameras,
                                 int rows,
                                 int cols,
                                 int *image_data);
us_image_post_xy_char_reading( STATUS_TYPE status,
                               TIME timestamp,
                               int num_cameras,
                               int rows,
                               int cols,
                               int *image_data);
us_image_post_byte_symbolic_reading( STATUS_TYPE status,
                                     TIME timestamp,
                                     int num_cameras,
                                     int rows,
                                     int cols,
                                     int *image_data);
us_image_post_threshold_reading( STATUS_TYPE status,
                                 TIME timestamp,
                                 int num_cameras,
                                 int rows,
                                 int cols,
                                 int *image_data);
us_image_post_spatial_derivative_reading( STATUS_TYPE status,
                                          TIME timestamp,
                                          int num_cameras,
                                          int rows,
                                          int cols,
                                          int *image_data);
```

```
us_image_post_temporal_derivative_reading( STATUS_TYPE status,
                                           TIME timestamp,
                                           int num_cameras,
                                           int rows,
                                           int cols,
                                           int *image_data);
us_image_post_recognition_reading( STATUS_TYPE status,
                                   TIME timestamp,
                                   int num_cameras,
                                   int rows,
                                   int cols,
                                   int *image_data);
us_image_post_range_reading( STATUS_TYPE status,
                             TIME timestamp,
                             int num_cameras,
                             int rows,
                             int cols,
                             int *image_data);
us_image_post_flow_reading( STATUS_TYPE status,
                            TIME  timestamp,
                            int num_cameras,
                            int rows,
                            int cols,
                            int *image_data);
us_pio_enable( int channel);
us_pio_disable( int channel);
us_pio_set_mode( int  direction);
us_pio_control_write( int  info,
                      int bits_data,
                      int  run,
                      int  ref );
us_pio_scale( int channel,
              double m ,
              double b);
us_pio_data_write( int  type,
                   union( double dvalue,
                          int    ivalue);
    );
us_pio_data_read( int  type,
                  int channel);
us_pio_bit_read( int channel_num,
                 int bit);
us_pio_bit_set( int channel_num,
                int bit);
us_pio_toggle_bit( int channel_num,
                   int bit);
us_pio_input_data( int  type,
                   union( unsigned long data_register,
                          double value);
                   unsigned long data_mask);
us_tlc_use_joint_reference_frame();
us_tlc_use_Cartesian_reference_frame();
us_tlc_use_representation_units( Measurement_units_type units);
us_tlc_use_absolute_positioning_mode();
us_tlc_relative_positioning();
us_tlc_use_wrist_positioning();
us_tlc_use_tool_positioning();
us_change_tool( int i);
us_tlc_use_modified_tool_length_offsets( int r);
us_tlc_use_normal_tool_length_offsets();
us_tlc_use_no_tool_length_offsets();
us_tlc_use_kinematic_ring();
us_tlc_start_manual_motion( AxisMask axis);
us_tlc_stop_manual_motion( AxisMask axis);
us_tlc_start_automatic_motion( AxisMask axis);
us_tlc_stop_automatic__motion( AxisMask axis);
```

```
us_tlc_start_traverse_motion();
us_tlc_stop_traverse_motion();
us_tlc_start_guarded_motion();
us_tlc_stop_guarded_motion();
us_tlc_start_compliant( AxisMask axis,
                        double *spring);
us_tlc_stop_compliant( AxisMask axis);
us_tlc_start_fine( AxisMask axis,
                   double errtolerance,
                   int proximity);
us_tlc_stop_fine( AxisMask axis);
us_tlc_start_move_until( AxisMask axis,
                         double *contact_forces);
us_tlc_stop_move_until( AxisMask axis);
us_tlc_start_standoff( AxisMask axis,
                       double *distance);
us_tlc_stop_standoff( AxisMask axis);
us_tlc_start_force_positioning();
us_tlc_stop_force_positioning();
us_tlc_use_dof( int dof);
us_load_cycle_time( double time);
us_tlc_load_representation_units( Measurement_units_type units);
us_tlc_load_length_units( Measurement_units_type units);
us_tlc_load_relative_positioning( TRANSFORM * t);
us_tlc_zero_relative_positioning();
us_tlc_zerot_program_origin( TRANSFORM * t);
us_tlc_load_kinematic_ring( Measurement_units_type units,
                            int  mask);
us_tlc_load_base_parameters( TRANSFORM * trBase);
us_tlc_load_tool_parameters( char name[128],
                             double dx,
                             dy,
                             dz,
                             double ux,
                             uy,
                             uz,
                             double normal_threshold,
                             double tangential_threshold,
                             ORIENTATION_TYPE heading);
us_tlc_load_object( OBJECT obj_id,
                    TRANSFORM * t);
us_tlc_load_object_base( TRANSFORM * t);
us_tlc_load_object_offset( int i,
                           TRANSFORM * t);
us_tlc_load_delta( int  delta,
                   double magnitude,
                   double frequency);
us_tlc_load_obstacle_volume( int i,
                             TRANSFORM * t);
us_tlc_load_blending( double dist,
                      BLEND_TYPE blend);
us_tlc_load_feed_rate( double feed_rate,
                       Measurement_units_type units);
us_tlc_load_traverse_rate( double traverse_rate,
                           Measurement_units_type units);
us_tlc_load_acceleration( double accel,
                          Measurement_units_type units);
us_tlc_load_jerk( double jerk,
                  Measurement_units_type units);
us_tlc_load_proximity( AxisMask axis,
                       double distance);
us_tlc_load_contact_forces( TRANSFORM * tr,
                            Representation_units_type units,
                            int dof,
                            long cfSelVect,
                            long cfComplyVect,
                            double *cfFtSetpoints,
                            double *cfFtGains,
                            double *cfMaxFcVel);
us_tlc_load_joint_limit( AxisMask axis,
                         double  *jtLimit);
us_tlc_load_contact_force_limit( double  *ctFLimit);
us_tlc_load_contact_torque_limit( double  *ctTLimit);
us_tlc_load_sensor_fusion_pos_limit( double *fsPLimit);
us_tlc_load_sensor_fusion_orient_limit( double *fsOLimit);
us_tlc_load_segment_time( double time);
us_tlc_load_termination_condition( int  condition,
                                   int select,
                                   double testTime,
                                   double endTime,
                                   double endTransDel,
                                   double endAngDel,
                                   double endTransVel,
                                   double endAngVel,
                                   double endForceErr,
                                   double endTorqueEff,
                                   double endForceVel,
                                   double endTorqueVel);
us_tlc_incr_velocity( int i,
                      double increment);
us_tlc_incr_acceleration( int i,
                          double increment);
us_tlc_set_goal_position( double *data);
us_tlc_goal_segment( SEGMENT_SELECT *segment);
us_tlc_adjust_axis( AxisMask axis,
                    int i,
                    double *value);
us_tlc_update_sensor_fusion( TRANSFORM * update);
us_tlc_select_plane( AxisMask  axis);
us_tlc_use_cutter_radius_compensation( double radius);
us_tlc_start_cutter_radius_compensation( double side);
us_tlc_stop_cutter_radius_compensation();
us_tlc_straight_traverse( double x,
                          double y,
                          double z);
us_tlc_arc_feed( AxisMask first_axis,
                 AxisMask second_axis,
                 double rotation,
                 double axis_end_point);
us_tlc_straight_feed( double x,
                      double y,
                      double z);
us_tlc_parametric_2d_curve_feed( FUNCTION_PTR f1,
                                 FUNCTION_PTR f2,
                                 double start_parameter_value,
                                 double end_parameter_value);
us_tlc_parametric_3d_curve_feed( FUNCTION_PTR xfcn,
                                 FUNCTION_PTR yfcn,
                                 FUNCTION_PTR zfcn,
                                 double start_parameter_value,
                                 double end_parameter_value);
us_tlc_nurbs_knot_vector( int i,
                          double k);
us_tlc_nurbs_control_point( int i,
                            double x,
                            double y,
                            double z,
                            double w);
us_tlc_nurbs_feed( double sStart,
                   double sEnd);
us_tlc_teleop_force_reflection( double *data);
us_tds_load_user( USER_TYPE user);
```

```
us_tds_select_program( char filename[128]);
us_tds_execute_program( char filename[128]);
us_tds_select_operation( int  task);
us_tds_select_opmode( int  type,
                      AxisMask axis);
us_tds_load_selections( int selection,
                        char name[128]);
us_tds_load_reference_units( Measurement_units_type units);
us_tds_load_rates( int  selection,
                   int  units,
                   double rate);
us_tds_load_origin( int  selection,
                    char name [128]);
us_tds_load_sensing( int  selection,
                     char  sensor_name[128],
                     Attribute_t attr,
                     double setting);
us_task_framework( int  task,
                   int step_number,
                   char macro_name[128],
                   USER_TYPE user,
                   int  type,
                   AxisMask axis,
                   int number_of_agents,
                   char  agent_class[128][100],
                   char  agent_list[128][100],
                   char tool_class[128],
                   char tool_name[128],
                   char object_class[128],
                   char object_name[128],
                   int task_units,
                   POSITION program_home,
                   POSITION program_origin,
                   POSITION relative_origin,
                   TRANSFORM    * base_frame,
                   TRANSFORM    * tool_frame,
                   TRANSFORM    * zero_axes_force,
                   TRANSFORM    * zero_tool_force,
                   int default_task_reference_units,
                   int task_reference_units,
                   double set_task_space_acceleration_limit,
                   double set_task_space_acceleration_time,
                   double feed_rate,
                   double feed_rate_units,
                   double traverse_rate,
                   int traverse_rate_units,
                   double default_force_setting,
                   double guarded_proximity_setting,
                   double viscosity_setting,
                   double humidity_setting,
                   double desired_temperature,
                   double temperature_limit,
                   double noise_limit);
us_tk_macro( char framework_file [128],
             char action_file [128],
             char plan[128]);
us_select_resource( TASK_ID tid,
                    RESOURCE_SELECT agent,
                    SUBSYSTEM_ID ssid,
                    int  type);
us_select_tool( TASK_ID tid,
                END_EFFECTOR_SELECT tool,
                SUBSYSTEM_ID ssid);
us_select_sensor( TASK_ID tid,
                  RESOURCE_SELECT agent,
                  SUBSYSTEM_ID ssid,
                 int  type);
us_interp_run_plan( SUBUSYSTEM_ID ssid,
                    int  type ,
                    char plan[128]);
us_interp_halt_plan( SUBUSYSTEM_ID ssid);
us_ptps_input_request( SUBUSYSTEM_ID ssblocker,
                       SUBUSYSTEM_ID ssenabler,
                       int  type);
us_ptps_output_enable( SUBUSYSTEM_ID ssblocker,
                       SUBUSYSTEM_ID ssenabler,
                       int  type);
us_tps_freespace();
us_tps_guardede();
us_tps_constact();
us_supervisory_mode();
us_select_feature( FEATURE surface,
                   double fx,
                   fy,
                   fz,
                   double fo1,
                   fo2,
                   fo3);
us_select_material( MATERIAL_TYPE m,
                    double maxx,
                    maxy,
                    maxz,
                    double minx,
                    miny,
                    minz,
                    double fo1,
                    fo2,
                    fo3,
                    double strength,
                    double minforce,
                    double maxforce);
us_load_obstacle( FEATURE obstacle);
us_load_pattern( GEOMETRY_PATTERN pattern);
us_tps_mark_event( int  event);
us_ptps_enable( int enable);
us_framework( char name [128]);
us_symbolic_item( char name [128]);
us_symbolic_item_attribute( char name [128],
                            char attribute_name[128],
                            int size,
                            int xdim,
                            int ydim,
                            Representation_units_type  rep,
                            Measurement_units_type   units,
                            generic_value_a values);
us_om_create( int  type,
             char name [128],
             char device[128],
             GEOMETRY data);
us_om_delete( int  type,
              char name [128]);
us_om_modify( int type,
              char name [128],
              char device[128],
              GEOMETRY data);
us_oc_calib( int type,
             char name [128],
             char device[128],
             GEOMETRY data);
us_oc_set_attr( char name [128],
                Modifier_t modifier,
                Attribute_t attributes,
                int size,
                Representation_units_type  rep,
                Measurement_units_type units,
                generic_value_a value);
us_oc_get_attr( char name [128],
                Modifier_t modifier,
                Attribute_t attributes);
us_ok_record( char name [128]);
us_ok_create( char name [128],
              OBJECT ob);
us_ok_delete( char name [128]);
us_ok_modify( int size,
              void * data);
us_ok_modify_attribute( Attribute_t attr,
                        int size,
                        void * data);
us_ok_attr_query( Attribute_t attr);
us_registered_id( char name [128]);
us_ok_attr_response( Attribute_t attr,
                     double  *values);
us_trd_open( char name[128],
             int  type);
us_trd_record( char name[128]);
us_trd_positioning( char name[128],
                    int num_element);
us_trd_name_item( char name[128]);
us_trd_delete_item();
us_trd_set_joint_mode( double dof);
us_trd_set_Cartesian_mode( double dof);
us_trd_modify( char name[128],
               int num_element,
               double *data);
us_trd_add_element( double *data);
us_sgd_error( char name [128],
              int obj_id1,
              int obj_id2,
              double x,
              y,
              z);
#endif
```

Index