

GadGETStackIP/OPC

Multi-Platform ANSI 709.1 (LonTalk®)
Protocol Stack with
EIA 852 IP Tunneling Capability and
OPC Server Support



360 West 920 North
Orem, Utah 84057-3042
USA
(voice) 801.226.7607
(fax) 801.226.7608

info@gadgettek.com
www.gadgettek.com

Document Version 2.1 - 2002.10.28



INTRODUCTION

GadgetStack™ is a software implementation of the EIA 709.1 Control Network Protocol Specification, the protocol also used by LONWORKS® devices. GadgetStack is interoperable with the built-in EIA 709.1 implementation contained in Neuron® Chips and allows designers to implement interoperable nodes on the target microprocessor of their choice. The goal behind the development of the GadgetStack software was to provide a highly portable cross platform implementation of the EIA 709.1 standard protocol.

Adept first authored under contract to Echelon Corp. the C Reference Implementation of the LonTalk protocol on the Motorola MPC68360. This C Reference implementation became the basis for the EIA 709.1 standard protocol. Adept has since developed a performance enhanced implementation and ported it to “little endian” platforms such as Intel processors running Windows, Linux, and QNX. GadgetStack also runs on Motorola PowerPC and ARM processors. Adept has developed a version of the stack for NASA and general aviation following the RTCA/DO 178B guidelines.

GadgetStackIP connects to the outside world through a TCP/IP interface. Together with an EIA-852 compliant tunneling gateway, GadgetStack applications can interact with nodes residing on a LonTalk network.

The 709.1 packets are tunneled over IP/ethernet to the gateway. In essence, ethernet transports packets from the link layer of the 709.1 stack running on the host to the 709.1 physical layer running on the router. This arrangement provides the highest degree of platform independence and portability. Virtually any platform with an IP stack and ethernet interface can support the GadgetStack. In the future, Adept plans to provide support for GadgetStack and GadgetNIC for even more processors and operating systems.

Figure 1 shows the generic architecture for the GadgetStack implementation. The GadgetStack

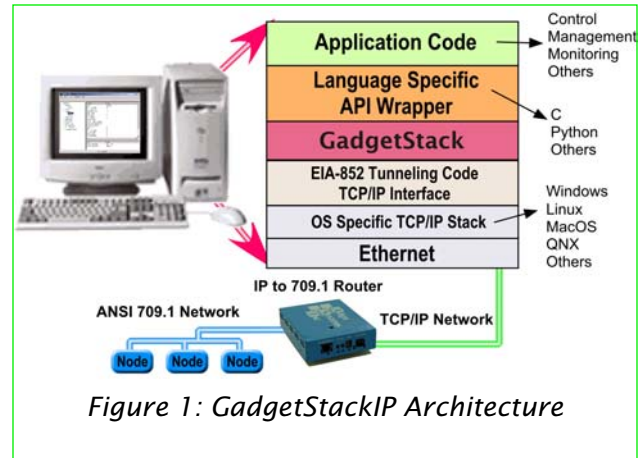


Figure 1: GadgetStackIP Architecture

FEATURES

- * Portable EIA 709.1 implementation
- * Multiple operating systems and processor platforms
 - ♦ Windows, Linux, and QNX
 - ♦ Little-endian and big-endian versions
- * Cross-platform scriptable rapid application development
- * Supports embedded network management and self-installation tools
- * TCP/IP Interface
- * EIA-852 IP Tunneling Support
- * OPC Server Interface
- * Convenient API
- * Scalable applications on IP backbones
- * Interoperable installation and configuration

can support multiple stacks running on one host or multiple hosts.

SUPERVISORY APPLICATIONS ON IP BACKBONES

GadgetStack provides the core capability a system's integrator needs to implement supervisory applications that connect over a high speed Ethernet or other IP backbone to a IP to LonTalk gateway. Traffic from multiple 709.1 channels can be tunneled over high speed IP media thereby enabling applications such as remote monitoring, logging, or traffic consolidation. Each GadgetStack can be configured to broadcast to or receive from multiple other IP gateways or GadgetStacks. This enables much larger scale installations and leverages the installed Ethernet or other TCP/IP based infrastructure. With GadgetStack based applications OEMs can maximize developer productivity by putting their applications on the most suitable and convenient platform. Adept is using GadgetStack for a set of network management tools to install and configure dependable topology and self-

healing networks for a survivable Naval ship-board automation infrastructure.

INTEROPERABLE CONVENIENT CONFIGURATION

The GadgetStack implementation interoperates with leading network management tools, such as, LonMaker. Networks variable binding is interoperable with all compliant implementations of the ANSI 709.1 standard. In addition, GadgetStack enables LonTalk over IP communications with an implementation of the new open EIA 852 standard specification for interoperable Ansi 709.1 to IP gateways and tunneling repeaters. The 852 library was designed to be compliant with existing gateway configuration software to make upgrades easy for system's integrators GadgetStack is designed for seamless integration into existing LonWorks© control networks and is highly interoperable with existing management tools, nodes, gateways, and routers.

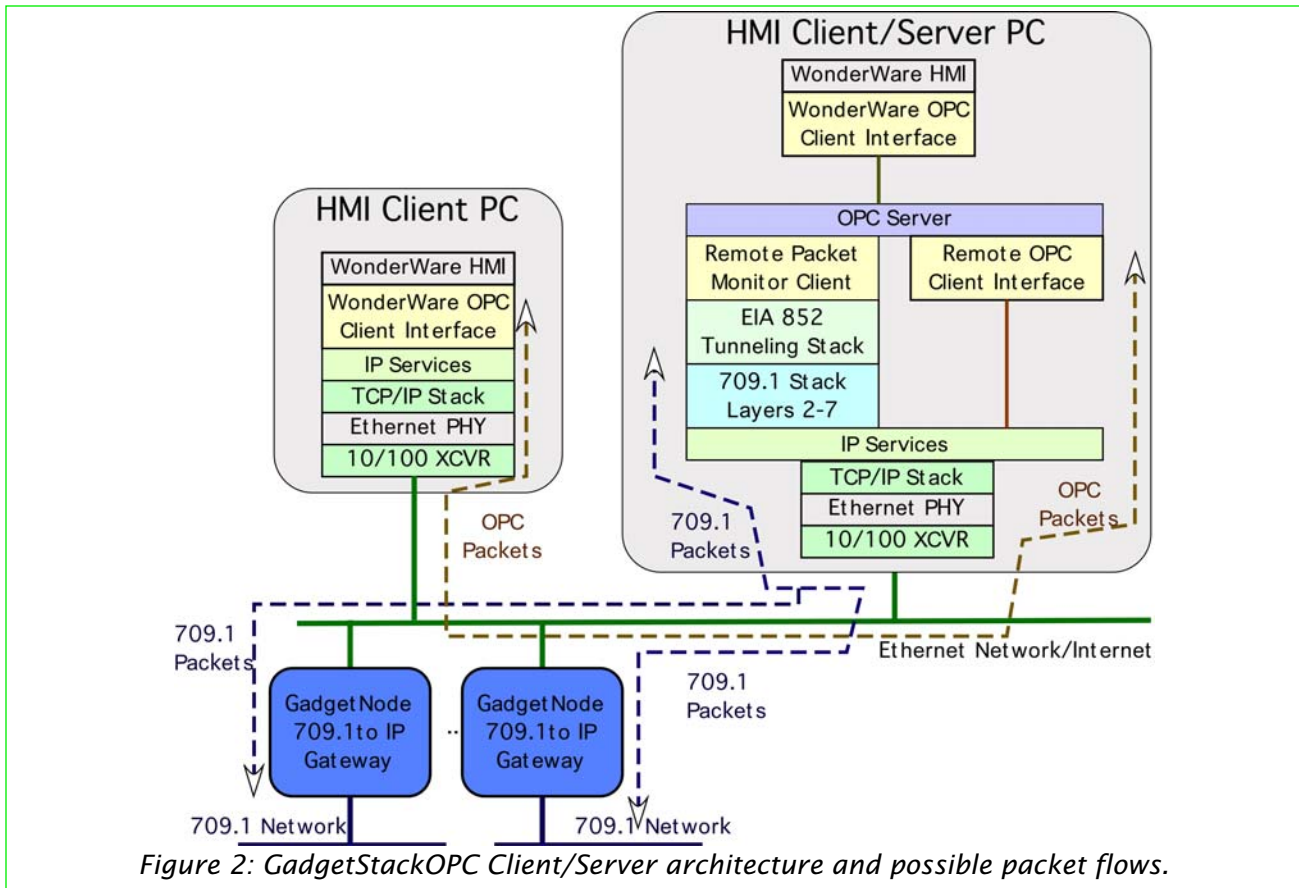


Figure 2: GadgetStackOPC Client/Server architecture and possible packet flows.

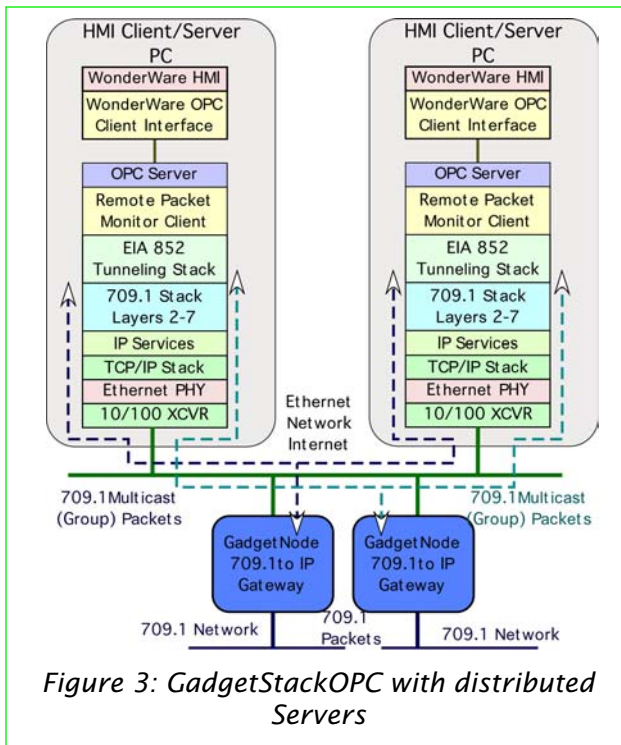


Figure 3: GadgetStackOPC with distributed Servers

GADGETSTACK OPC SERVER

GadgetStack now comes with an OPC (Open Process Control, originally OLE for Process Control) server interface. This allows OEMs to attach custom HMIs as OPC clients. This brings all the power and performance of GadgetStackIP and IP tunneling, and GadgetGateway connectivity to supervisory and control applications and human machine interfaces. Multiple HMIs may be clients of one OPC server or distributed servers may be used where each PC may have a dedicated OPC server and client. These configurations are shown in Figures 2 and 3. This enables redundant HMIs and avoids single point failures for critical control applications.

GADGETSTACK API

The GadgetStack protocol stack is written in “C”. The C language API (Application Program Interface) is designed to mimic the Neuron C API to make it easy for developers to port applications. Because the stack is not limited by the size of a Neuron chip’s memory map, developers are able to tailor node memory requirements to their needs.

The GadgetStack API provides C prototypes to functions in two forms.

1. The Adept C prototype which Adept will continue to evolve as future features are incorporated into the GadgetStack and related ANSI/EIA 709.1 standards.

2. C prototypes which are intended to maintain backward compatibility with Echelon’s Neuron C, and which represent the Service Interface documented in section 11.3 of the ANSI/EIA 709.1 standard.

The following files are typically included in the GadgetStack distribution:

api.h, lontalk.h, and custom.h

Api.h is the application program interface include file. It contains the ANSI 709.1 address typedefs, 709.1 message structure typedefs, GadgetStack network variable type definitions and the GadgetStack API function prototypes. Lontalk.h contains ANSI 709.1 enumerated constants as well as source and destination address type definitions. Custom.h is the custom.c include file. It contains the data types and definitions that can be modified to build a custom 709.1 compliant node.

The following declarations define the API interface. These declarations are located in api.h:

```
extern MsgIn msgIn; // Incoming message data
extern MsgOut msgOut; // Outgoing message data
extern RespIn respIn; // Incoming response data
extern RespOut respOut; // Outgoing response data
extern Boolean msgReceive; // True if valid data is in
msgIn
void msgFree(); // "Frees" data in msgIn
Boolean msgAlloc(); // Is non-priority msgOut available
Boolean msgAllocPriority(); // Is priority msgOut available
void msgCancel(); // Cancels msgAlloc or msgAllocPriority
void msgSend(); // Sends msgOut
extern Boolean respReceive; // True if valid data in
respIn
void respFree(); // "Frees" data in respIn
Boolean respAlloc(); // Is respOut available
void respCancel(); // Cancels respAlloc
void respSend(); // Sends respOut
void doApp(void); // User defined
void resetApp(void); // User defined
void msgCompletes(MsgStat stat, MsgTag tag); // User
defined
typedef enum { MSG_FAILED, MSG_SUCCEEDED, MSG_COMPLETED
} MsgStat;
```

The API includes some utility functions such as,

```
int16 AddNV(NVDefinition* NVdef);
void Propagate(void);
void PropagateNV(int16 nvIndex);
void PropagateArrayNV(int16 arrayNVIndex, int16 nvIndex);
void Poll(void);
void PollNV(int16 nvIndex);
void PollArrayNV(int16 arrayNVIndex, int16 nvIndex);
void GoOffline(void);
void GoUnconfigured(void);
MsgTag NewMsgTag(BindNoBind bindStatusIn);
Boolean ServicePinMessage(void);
void SetMsTimer(MsTimer *timerOut, uint16 initValueIn);
```

Some example code using the C language API is shown below.

```

/* Example Code */
void DoApp(void)
{
    int i;

    if (MsTimerExpired(&lightTimer))
    {
        LightOff();
    }

    /* IOChanges should be called before checking the
    status of io pins */
    if (IOChanges(0) && gp->ioInputPin0)
    {
        for (i = 0; i < 10; i++)
        {
            if (msg_alloc_priority())
            {
                msg_out.priority_on = TRUE;
                msg_out.tag         = tag1;
                msg_out.code        = 1;
                msg_out.data[0]     = 5; /* Some data */
                msg_out.len        = 1;
                msg_out.authenticated = FALSE;
                msg_out.service     = UNACKD;
                /* gp->msgOut.addr.snode = snodeAddr; */
                msg_send();
            }
        }
        intOut++;
        PropagateNV(intOutIndex);

        /* Update whole array */
        intArrayOut[0]++;
        intArrayOut[1]++;
        intArrayOut[2]++;
        PropagateNV(intArrayOutIndex);
    }

    /* Ignore incoming messages */
    msg_receive();

    /* Ignore incoming responses */
    resp_receive();
}

Status AppInit(void)
{
    /* Register Network variables */
    intOutIndex      = AddNV(&intOutDef);
    longOutIndex     = AddNV(&longOutDef);
    intArrayOutIndex = AddNV(&intArrayOutDef);
    intInIndex       = AddNV(&intInDef);
    longInIndex      = AddNV(&longInDef);
    intArrayInIndex  = AddNV(&intArrayInDef);

    /* Make sure we were successful in registering all
    variables */
    if (intOutIndex == -1 ||
        longOutIndex == -1 ||
        intArrayOutIndex == -1 ||
        intInIndex == -1 ||
        longInIndex == -1 ||
        intArrayInIndex == -1)
    {
        return (FAILURE);
    }

    tag0 = NewMsgTag (BINDABLE);
    tag1 = NewMsgTag (BINDABLE);

    /* Make sure we got the tags successfully */
    if (tag0 == -1 || tag1 == -1)
    {
        return (FAILURE);
    }
}

```

```

}

return (SUCCESS);
}

```

PYTHON API

Adept is also developing a 4th generation scripting language API using the open source Python language. This will be highly portable. The future Python API will make it easy to rapidly develop complex applications and user interfaces such as embedded network management tools or self installation tools. Because Python is embeddable and extensible, it works well for both IT server side applications and embedded system control applications. Other languages can be supported as well.

CUSTOMIZED VERSIONS

Adept provides custom solutions for interoperable network-centric automation systems to OEMS with large quantity applications. Customized enhancements to the GadgetStack and GadgetGateway can be provided such as support for hot fail over. If you have need for custom features please call.

ORDERING

The price for the either the GadgetStackIP 2.0 Developer Kit or the GadgetStack OPC Beta Developer is US \$2495.00. This includes binary libraries, headers files, API, sample application source code and phone support for installation. Steeply discounted runtime licenses are available depending on platform, application, and quantity. Additional support contracts are available, as well as on-site support.

Prices are subject to change without notice. For up-to-date ordering, pricing, and availability information, please contact our sales staff.

801.226.7607 (voice)

801.226.7608 (fax)

info@gadgettek.com (E-mail)

<http://www.gadgettek.com/>

WonderWare trademark of WonderWare Corporation. Echelon, Neuron, LonTalk, LNS, and LonWorks are trademarks of Echelon Corporation. All other trademarks are the property of their respective owners.