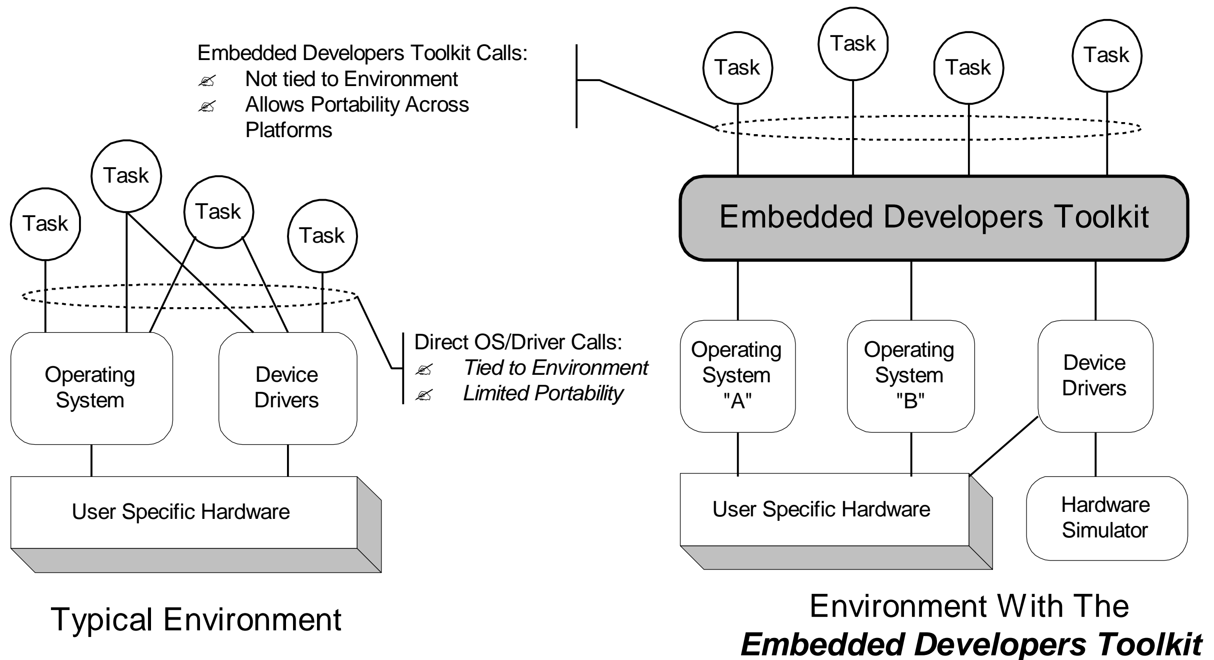
Embedded Developers Toolkit (EDT)
Product Description

The Embedded Developers Toolkit provides a set of Operating System Independent host and target tools which promote consistency in design, allow rapid prototyping of the users application code, and host based simulation and portability across different Operating Systems. This toolkit allows a several month head start on any embedded application, and is ideal for situations where time to market is critical.

The EDT is effectively an "interface" layer to the underlying Operating System and hardware platform. This interface allows the user to easily move between environments, during development, as well as over the life cycle of the users product. Depicted below are two environments; a typical development environment, and a development environment which uses the EDT:



In a typical environment, the users Application Code makes direct calls to the Operating System, and has extremely tight ties to the underlying hardware, which is most often a custom design. Frequently the developer faces the problem of how to perform development of the application prior to the availability of working hardware.

The toolkit "abstracts" the users Application Code from the underlying environment. This allows the developer an easy path to build prototype environments, allowing Application development prior to arrival of the production hardware, as well as portability across Operating Systems (including a host based development environment).

Additionally, the EDT provides the users with the option of using a time tested methodology for architecting their embedded application. The methods supported by the EDT promote consistency in design, ease of implementation, ease of debug, and most important, easy maintenance of a system over time – seasoned developers who have faced over-complicated designs in the past understand the importance of building a system which is simple to maintain over time and changes in staff.

Copyright © 1999-2002 – ICF Ventures, Inc

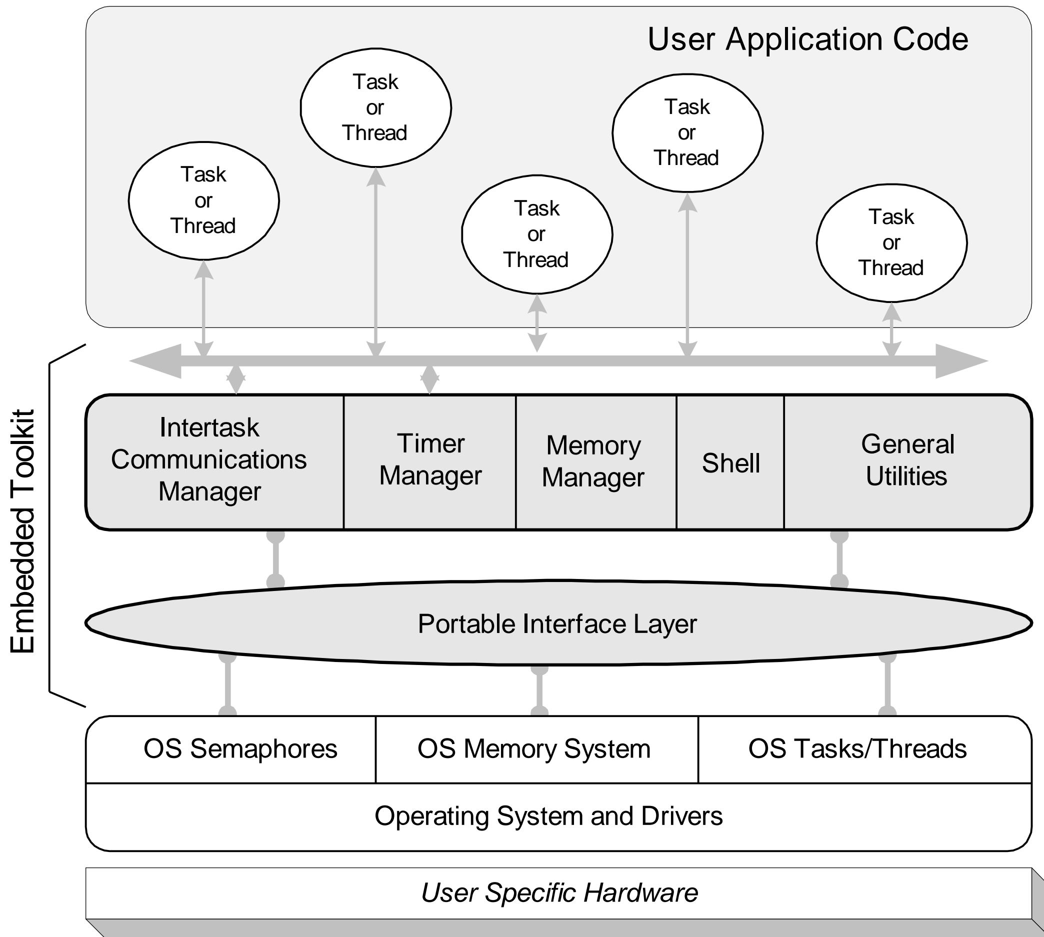
In achieving ideals such as these, developers are faced with developing their own interface layer, custom to their project and specific to their needs. This effort is often several weeks in the making, and with implementation, must still be debugged and finalized. In this day, when “time to market” is critical, the EDT allows the engineer the ability to put their time and efforts into their own application, not into necessary interface layers.

Target Utilities

The target based utilities provides the following key elements to an application:

1. Operating System Interface Layer.
2. A Intertask/Interprocess Communications Manager (ICM) which provides consistent messaging between application tasks and processes, hiding the Operating System specific mechanisms.
3. A Timer Manager (TM) for managing timing functions in the Embedded Application.
4. A Shell (Linux only) which supports dynamic interface with the users application, including the ability to interface with the ICM, TM, display system status, dynamically load application object modules and call user procedures from the command line.
5. A Logging mechanism for handling system diagnostic messages.
6. A Memory Manager which allows easy tracking of allocated memory blocks, and in the Linux environment, supports shared memory constructs between processes.
7. A set of general utilities, such as generic linked list handlers.

These target utilities, shown below, effectively provide an interface layer which isolates the users Application Code from the underlying Operating System and hardware environment.



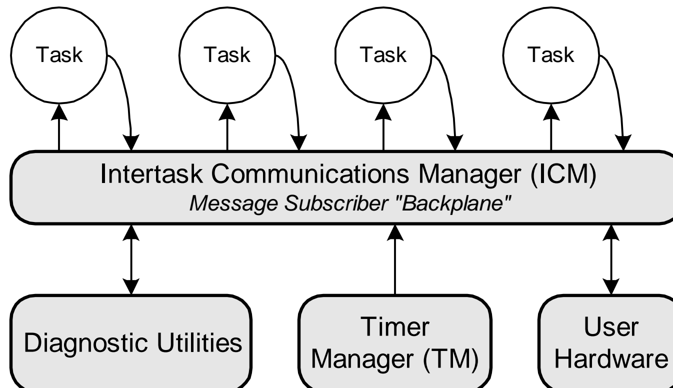
This target environment promotes the following design methodologies:

1. Consistency in Task/Process design; all Tasks/Processes bodies start from a standardized template. In the Linux environment, support is provided for both threads and heavy processes.
2. Consistent Intertask/Process Communications; all Tasks/Processes communicate using a standardized message format. In the Linux environment, support is provided to move messages both between threads, and between processes.
3. Distributed Message Delivery; any messages routed through the local ICM may be routed transparently over a IP based network to an ICM on other machines. Both TCP and UDP are supported.
4. Consistent Time Management; all system Timers behave identically, and work in conjunction with the ICM – when a "Time" event elapses, the Task/Process is alerted through a standard ICM Message.
5. Consistent Diagnostic Messages; instead of the usual `printf()` function, an infrastructure is provided which allows all diagnostic messages to be displayed, logged or disabled by Task/Process ID, or by Diagnostic Message Type.
6. Tracking of memory leaks within the system.
7. Fully interactive queries of the System Status; in the Linux environment, this is done through the EDT Shell, while in the VxWorks environment, done through the VxWorks shell.

Through these key methodologies, the developer is able to rapidly prototype a system environment that programmers of all skill levels are able to work within.

Target Utilities Details

The core of the Target Utilities is based on the Intertask Communications Manager (ICM). The ICM promotes consistent messaging between tasks, allowing all tasks to be derived from a common template. All tasks (or processes in a Linux environment) communicate through the ICM, as shown below:



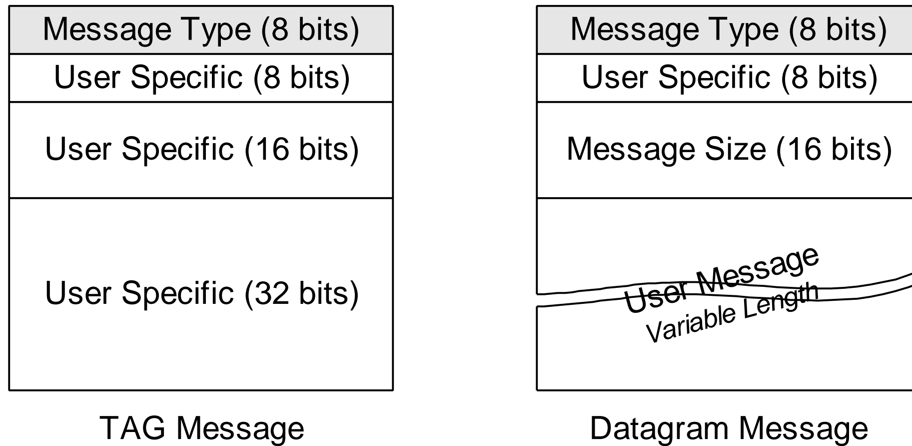
Intertask Message Subscriber Model

Since each task (or process) is derived from a common template, every module in the system operates identically: they each receive messages through the ICM, and as they go through state changes, generate messages which are routed through the ICM.

Diagnostic tools allow the user to monitor message flow in the system during development and integration, as well as to "inject" messages to the system (causing state changes). With careful design practices, the user hardware is "hidden" through this layer, allowing initial development on the host.

The Timer Manager allows the user to cause events based on time to be sent into the system through standard messaging, which again promotes consistency in design, as well as easy of integration (when trying to debug a system, any message can be generated at any time, including Timer messages, which allows the developer to easily transition the system from state to state).

Two types of messages are supported by the ICM; TAG Messages, and Datagrams, as shown below:

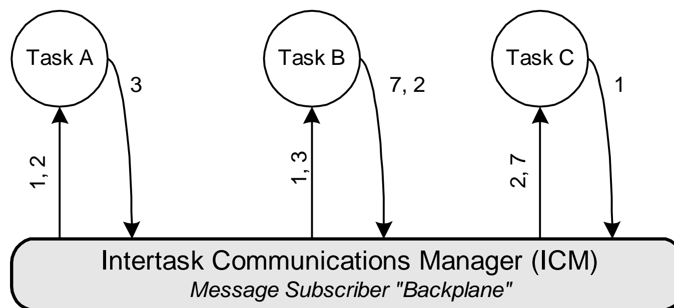


TAG Messages are short messages routed by the ICM directly to a task or process; these messages require no memory allocation or management.

Datagram Messages are blocks of memory containing messages which can be of variable length. A pointer to this message is routed by the ICM to a task or process, and the ICM provides utilities to manage the memory for these messages. Application code acquires memory from the ICM for a message, fills it in with application specific data, and then routes it through the ICM. The receiving task or process receives the message, processes it, and then releases the memory. This operation is similar to a `malloc()` and `free()` type operation, with the added advantage that the developer can always monitor the state of these memory blocks through the EDT utilities.

TAG Messages are generally used to cause a state change in a system, or alert the application to a pending interrupt, while Datagrams will contain more detailed information. In a Linux environment, where Processes may be isolated from other Processes through the Memory Management system, the ICM takes care of copying the messages between memory spaces. This frees the developer from having to attach shared memory, or manage multi-byte messages through traditional constructs such as queues.

Message routing is done through a “subscriber” model, as shown below:



Message Routing

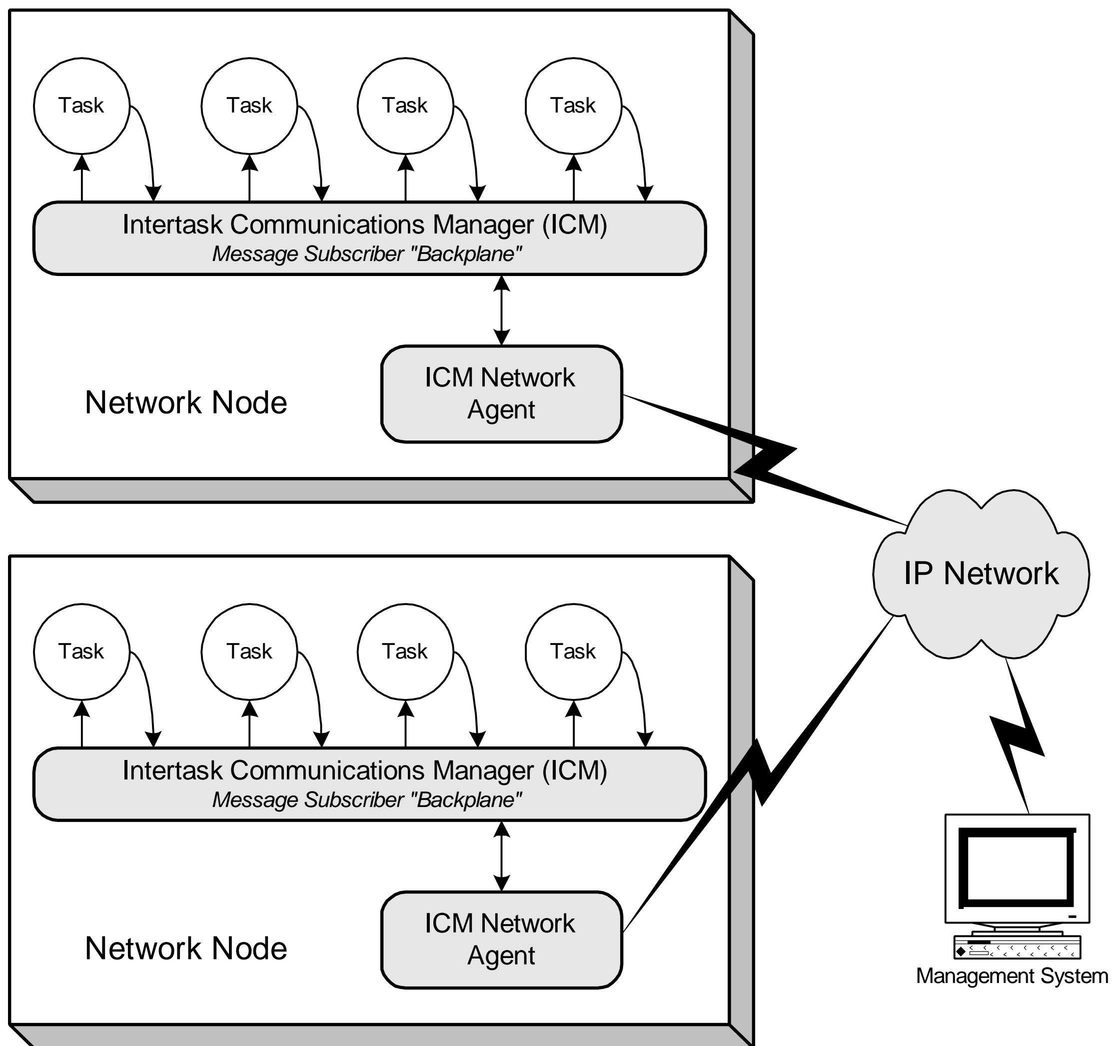
Each task or process in a system “subscribes” to messages. The gray area depicted in the diagrams of the two message types represents a “Message Type.” All messages in a system have a unique identifier (a number).

Copyright © 1999-2002 – ICF Ventures, Inc

Message “subscription” is based on a task or process “subscribing” to Message Types. In the example shown above, Task A has Subscribed to Message Types 1 and 2, Task B requests Message Types 1 and 3, and Task C requires Message Types 2 and 7.

As shown in this example, Task A generates Message Type 3; this message, when routed through the ICM, is delivered to Task B. Task B generates Message Types 7 and 2, and when routed, are delivered to both Task A and Task C . Task C generates Message Type 1, and this message, when routed through the ICM, is delivered to Tasks A and B.

Another extremely powerful capability of the EDT and ICM is Distributed Network Message Routing. As shown below, a Network Agent exists which allows ICM Messages to be routed and dispatched on remote nodes residing on an IP Network. Any ICM Message can be transmitted on an IP Network containing one or more nodes, without the need for the developer to design, implement and debug custom application protocols. Routing can be done using TCP or UDP, on a message by message basis.



Network Message Subscriber Model

Copyright © 1999-2002 – ICF Ventures, Inc

Through this “Subscriber” methodology, four key capabilities are achieved:

1. Messages are generated without need of one task to know the message destination; when developing the application, any given task in a system is not required to know a recipient. The recipient may not even exist, or could exist simply as a “stub task” derived from the template.
2. Transparent routing over IP networks; this routing can be between any number of nodes on a network. UNIX/Linux agents exist for allowing communications between target embedded nodes and host management machines.
3. Messages between tasks can easily be monitored through the EDT tools; when debugging, the developer can simply “enable” all messages, based on type, to be monitored via the EDT Diagnostic Monitoring tools – the user can easily see what is happening between tasks.
4. Any message can be generated from the EDT Shell at any time, allowing simple testing of a system.

Although this model does require that a methodology be followed, all State Driven Embedded Systems require a scheme for moving messages. The EDT gives the user a significant time and schedule advantage on their project by taking care of these details, allowing them to concentrate on their specific application.

Another important aspect of this concept and methodology is the management of “time” within a system. Applications and tasks will often initiate some action which requires a state change in the system (which may not occur due to external needs, such as incorrect or missing hardware responses). Additionally, systems generally require modules to execute periodically. The Timer Manager in the EDT is fully integrated with the ICM; tasks are able to “create” timers and then when starting them, ask the Timer Manager to deliver a application specific message (TAG or Datagram) when the timer expires. Through this, “time” becomes simply another message in the system, not a custom designed interface or module.

Most important in this entire concept and methodology is *consistency and simplicity*: all tasks behave the same. The user can monitor the system behavior without having to use debuggers, and the user can easily interact with their system during integration.

Host Tools

The Host Tools are designed to free the developer from spending significant amounts of effort setting up their development environment, and they help provide online documentation of the developers system. These tools are centered around three key components:

1. **Makefile Templates**. These templates are designed to work with the GNU tools, and provide generic makefile capabilities including automatic dependency generation. They work with all popular Configuration Management Tools, such as RCS, CVS and ClearCase.
2. **Automatic Documentation Generator**. This tool allows the user to follow a simple set of rules for file format and procedure and data structure headers. The source code is processed through this tool, and online HTML documentation is generated, allowing current documentation for the application code to be “built” from the “make” environment. All HTML pages are crossed linked following simple rules, allowing the user to easily browse both the auto-generated documentation, as well as the source code.
3. **S-Record and Binary Generator**. This tool takes the resultant linked object module and generates either S-Records or Binary Images suitable for installing in ROM.

Existing Ports and Porting

The EDT currently runs in both the VxWorks and Linux environments, using the GNU tools. Porting to a new environment is a straight-forward process that requires the following mechanisms from the Operating System:

1. Tasks and/or Processes.
2. Semaphores.

Copyright © 1999-2002 – ICF Ventures, Inc

3. Optionally, simple memory management/memory allocation support. The EDT provides utilties to manage memory, so the need for `malloc()` and `free()` is optional.

When using the GNU environment, the EDT easily ports to other environments, and porting can be done either by the user, or through contract with The Besemer Corporation.

Notable is the freedom from relying upon complicated ITC mechanisms; the underlying Operating System does not need to be full-featured. This simplified approach to using an Operating System is why the EDT ports easily to different environments, and allows the user to easily set up simulation/development environments.

Training and Support

Multiple options exist for training and support:

1. On-site training for a clients engineering and management staff. Training of this nature can help the client install the tools, and get their development environment configured and operational.
2. Training provided at our facilities, using our equipment.
3. Working with a clients staff to help architect their system, including development of System Requirements.
4. Working with a clients staff to implement and deploy their first EDT based application.

This training and support is tailored to each clients specific needs. To learn more about how this can benefit your organization, contact us for details.

Contact Us

For additional information of this product, contact Thomas E. Besemer at Thomas Besemer Consulting: http://www.tbcorp.com, or call (310) 822 5031.

Call concepts and information contained in this document are copyright © 1999-2002 ICF Ventures, Inc.