RADVISION
the V²oIP™ experts

H.323 Developer's Zone
Getting Started

Understanding The Developer's Zone

Deliver it fast! Provide the best quality! Remember time to market is critical! Product bugs were just reported- fix them ASAP! What can we do to improve the performance, memory consumption, and footprint? Are we interoperable with leading vendors? Do you know how to implement those new cool standard features, which were just ratified? What will put our product ahead of the rest?

Sound familiar?

Throughout the VoIP development life cycle, developers are under intense pressure from sales and marketing to resolve product delivery and maintenance issues.

Since early 1997, RADVISION has provided its award-winning H.323 Protocol Toolkit to hundreds of vendors worldwide. The H.323 Protocol Toolkit has been implemented in a variety of products including IP phones, advanced videoconferencing systems, PBXs, soft switches, gateways, gatekeepers, management systems and VoIP chipsets solutions.

This document is based on our experience in H.323 toolkit development and support since early 1997 when we had our first H.323 toolkit version shipped. This document can be used as a VoIP developer's checklist and guidelines from the pre-development stage and throughout the development and maintenance lifecycle. The document provides guidelines to optimize R&D productivity, maximize protocol capabilities, and handle the development and maintenance cycle efficiently.

This document reviews the following development topics:

- Documentation
- Toolkit tuning and configurations
- Application build file
- Reusing the sample application
- Using the ASN.1 compiler
- Code compilation issues
- OS Porting issues
- Developing VoIP products on a chipset
- Bug Handling
- Unit Testing Phase
- Product QA testing phase

Documentation

Toolkit documentation should be reviewed prior to any design or development phases. Product documentation is also essential for utilizing the toolkit correctly and efficiently through all development and maintenance cycles. Since H.323 is a protocol rich in services with many sub-protocols annexes and versions, comprehensive and quality documentation is essential for development and maintenance phases. We have identified four types of documents that should be included in a protocol toolkit:

- General Product Overview

  The General Product Overview should be a general overview of the capabilities of the toolkit and the related standards. It is important for introducing the developer, R&D team and the product manager. The Product Overview should include descriptions of the architecture, concepts, terms and practical examples of useful applications.

- Programmer's Guide

  The Programmer's Guide should describe the steps needed to develop an application and implement various modules/components of the toolkit as well as explain important features such as existing libraries and compilation and linking flags. By reading the Programmer's Guide before development, the developer can get a clearer picture of the toolkit's capabilities and features.

- API Reference Guide

  Once familiar with the overall concepts of toolkit, the developer can use the API Reference Guide to receive more detailed information about specific functions for the application. In the design and development phases, the API Reference Guide becomes the main source of information about the function details, usage and relationships with each other. The API Reference Guide should contain:

  - Function lists
  - Function processing purpose and logic
  - Formal function parameters definition
  - For each function parameter: default values, direction (In, Out, In-Out) and meaning
  - Returned values and meaning
  - Error codes and meaning

Toolkit tuning and configurations

H.323 toolkits should provide the necessary flexibility adjusting memory consumption, performance efficiency and timeout profiles based on product requirements.

Code Footprint

Code footprint size is related to the libraries used in the application and other setups. The toolkit should contain compilation flags, which control its code size (e.g. not using logging to reduce footprint, not using H.450, etc.).

Initial Memory Allocation

Initial memory allocation is defined as the fixed amount of memory that the protocol toolkit needs to allocate without depending on the amount of calls supported.
The amount of memory will stay the same regardless of whether the toolkit supports a single call or several thousand calls. Therefore, servers supporting a large number of concurrent calls will not be as sensitive to these figures as compact devices with limited memory resources.

Call Session Memory Consumption

There are two common approaches for memory allocation in almost any software development. One approach favors dynamic memory allocation for memory allocation on an as needed basis. Another approach is static memory allocation, which requires that all memory allocation be done at the startup/initialization time.

The main advantage of the dynamic memory allocation approach is that the application will never run out of memory as long as there is free memory on the machine. However, it requires more CPU processing and slows down performance. Therefore, static memory allocation might be a more reasonable solution for many developments.

There are several factors to check regarding toolkit memory allocation capabilities and options:

- Does the toolkit allow for configuration of maximum number of calls or does it require knowledge of internal memory structures?
- If it is possible to configure the maximum number of calls, can the developer specifically override a certain parameter?
- Does the toolkit manage the memory on a per call/channel basis or is it a pool of memory?
- How does the toolkit handle an incoming message larger than the allocated buffer? Will it recover from this situation?
- What monitoring capabilities does the toolkit provide? Is there a watchdog mechanism that enables monitoring of actual memory consumption in runtime and provides information for debugging in case of failure?

Timeouts Tuning

Tuning timeouts properly is another issue to be considered. The toolkit should provide default timeouts that comply with the standard recommendations if they are defined. In addition, the toolkit should provide the developer the ability to override the default timeouts if necessary. Overriding the default timeouts is important for developing more flexible applications based of targeted operational profile. Another more flexible option is that the toolkit enables setup timeouts during run-time. This will allow the application to control timeouts through an API according to programmed policies for operational scenarios (e.g. large timeouts due to high load or network degradation).

Maximizing Performance

Since performance is often an important gradient of the product value, it is important to understand how to configure the toolkit to optimize performance.

Message retransmission affects performance. The toolkit should be capable of discarding retransmitted RAS messages by only partial decoding; full message decoding will reduce performance. The same holds true for outgoing retransmission. The toolkit can help to avoid re-encoding the message by saving it for the required amount of time.

The way the toolkit stores outgoing and incoming messages and the interface used to browse and modify them is important for performance and functionality. The developer should have full flexibility to retrieve and edit message information but searching according to string comparison is time consuming. Therefore, the toolkit should also provide an interface for efficient message manipulation.

Another important factor is the H.323 procedures supported by the developed product. For example supporting FastStart or Q.931 multiplexing improves the application's performance of calls per second (CPS) per a given hardware configuration. Specifically, Q.931 multiplexing enhances performance in high load scenarios by several times.

Application Build File

The written code, operating system APIs, toolkit APIs referred to in a build file defines the application. Developers need to define the best build file and its compilation flags for the desired application. RADVISION suggests creating a file, which keeps all of the defined flags for later reference and reuse. It is probably a good idea to remove old object files from project spaces before rebuilding an updated application version in order to prevent compilation errors or cause wrong functionality. It is recommended that developers check the validity of variables and pointers, and return error and create log information in problematic cases. However, it is very important to define how it happens and to collect and present all the data that can help define the root of the problem. Sometimes it takes a long time before the bug can be discovered so it pays to create a mechanism that permits changing the debug level and specific logger options during the runtime of the system. There will be no need to restart the system and wait until the bug occurs again.

Reusing the Sample Application

To simplify and accelerate the development phase, it is important to have a sample application from the H.323 toolkit vendor. Using the sample application source code and understanding the process of the sample application source code can greatly reduce the time required in understanding toolkit usage. In addition, re-using the sample application source code to generate the desired application code provides the following benefits:

- Shortens the learning curve
- Reduces development time
- Reduces risk of bugs
- Makes the developed code more readable

The ASN.1 Compiler

For given ASN.1 definitions, the toolkit's ASN.1 compiler generates the proper Packed Encoding Rules (PER) objects that will be used by the H.323 toolkit APIs. The use of ASN.1 definitions and PER in the H.323 standards makes the messages passing thru the network a lot smaller than text but increases complexity of encoding and decoding them.

The ASN.1 compiler should provide the following capabilities:

- Minimal code size of the compiled ASN.1 definitions. This is very important for applications on a chip with limited memory resources.
- Minimal memory consumption of messages. ASN.1 definitions make it hard to have a static representation of messages in memory. They are sometimes recursive, with arrays of unlimited size and a lot of optional fields. The ASN.1 package should consume minimal memory resources for the messages when they are in their decoded form.
- Fast encoding/decoding operations. The speed of encoding and decoding messages is important for large applications. The speed here is also one of the issues that must be taken into account when creating the compiled ASN.1 definitions and when choosing the memory representation of the messages.
- Easy manipulation of message fields using generic API functions. Since ASN.1 messages can be represented as complex tree structures, using generic functions to access the fields makes it easier in the long run to write specific code that accesses the various fields within the messages.
- Multiple operating system support is also very important feature for an ASN.1 compiler. When moving from a big-endian platform to a little-endian platform such as Solaris to Win32, there is a risk of having some problems with the ASN.1 compiler or the ASN.1 encoder/decoder. Developers need to determine if the ASN.1 package needs to support both platforms or allow for a smooth migration between them. Therefore, an ASN.1 compiler that supports PER for both big-endian and little-endian platforms is beneficial.

OS Porting Issues

It is very important to have a portable toolkit that has an OS dependant “thin layer” of all OS dependant services such as I/O, timers and thread handling, with the rest of the toolkit source code using that layer. That “thin layer” should supply the toolkit with low-level OS dependant services and serve as an abstract layer between the H.323 protocol toolkit and the actual operating system used. Such a layer should provide support for multi-threaded and single-threaded applications, a clear and clean interface for porting, and a generalized use of networking interfaces of various operating systems. It is especially important in communication toolkits, to give special consideration to the toolkit’s network interface as it might be difficult to face a large number of clients with high packet rates. To meet these kinds of demands, designers of networking software have several options depending on the scenario. For example, in a multi-threaded environment, one can set each thread to handle several optimized amounts of clients with a non-blocking I/O while using ‘select()’ function, and if the platform is UNIX, then ‘poll()’ or ‘/dev/poll’ are optimized even more. It is important to notice that while using several tasks, synchronization is a must and semaphores/mutex should be controlled. There are many valuable optimized implementations of those controls on some embedded operating systems.

In conclusion, to maximize the H.323 toolkit’s OS portability, it is important to verify what options the toolkit vendor provides which meet the major short and long-term criteria. Additionally, when developing the application, all OS dependant calls should be made from a separate “thin layer” as described or the toolkit’s “thin layer”.

Developing VoIP Products on Chipsets

Products developed on chipsets reduce costs and quicken mass-market production. Application development time and integration costs can be greatly reduced by using an existing software framework that provides call signaling, call control and media transport functionalities. In order to develop VoIP products on chipsets efficiently, there are several guidelines to follow when considering the short and long-term issues of VoIP chipset development including:

- Multiple VoIP protocol support for H.323, SIP, MEGACO, and MGCP for single or multi-protocol solutions. More and more vendors of various types of VoIP products find that they need to support multiple protocols.
- An abstraction layer interface, tailored above a lower-level interface accessing DSP functions, for simplifying application development.
- Multiple RTOS support for platforms such as VxWorks, pSOS, Nucleus, OSE, to avoid dependency on a specific vendor.
- A Call Control module for SIP and H.323 and a Media Control module for MEGACO and MGCP.
- Transport modules such as an RTP toolkit and advanced features such as an SCTP toolkit. These modules should be flexible since some developers prefer to use their own tightly coupled implementations.
- Complete functional reference applications should be provided as a starting point for development.

Bug Handling

In order to find the root cause of a bug, the toolkit should contain tracing and debugging auxiliaries for setting up detailed logs explaining the log file's contents. Additionally, setting watchdog tracing services is important to know more about the state of resource consumption during the traced operation. It is essential to determine which events to trace, the level of detail required for tracing, setup factors the log file generation, methodology for reviewing the log files, and correlation of the log files with other tracing techniques.

However, when the application is running in "release" mode it should be tuned to have minimal or no debugging trace to minimize performance degradation.

Log File Settings

Log file settings define the log file detail levels and focus on tracing certain functionality types. Proper settings for debugging needs can prevent huge, redundant information and unreadable log file generation.

The toolkit should provide several levels of debugging including:

- No debugging
- Partial debugging (Several options should be available)
- Debugging specific modules for specific types of messages (only errors for example)
- Full debugging

Understanding Log File Results

The log file results include key word data type tags to simplify searching the log file for specific information. It is important to examine each function call parameter value and the returned value(s) generated by the function processing. The log file is logically structured at several activity tracing depth levels so it is important to understand the meaning of each depth level and the relationships between them.

Watchdog Services

The possibility to have a printed snapshot of memory usage can often help to analyze the cause of a bug. A few memory consumption parameters which should be analyzed include the current usage, the maximum allowed usage, and the maximum usage during the traced processing stages.

Unit Testing

The unit testing should be performed during the development phase when a few defined functionalities are completed and can be tested. For example, when the code can respond to certain messages or events. It is highly recommended to perform these tests as early as possible to detect bugs and recover from them before starting more advanced integration phases.
The toolkit should contain a short test application to perform automated unit tests on a per message level.

The following setups should be enabled:

- Message selection and sequence
- Message method
  (e.g. Fast-Start, RAS/no RAS, Tunneled H.245, Q.931 multiplexing, etc.)
- Message parameter setup
- Timeout defaults and specific setups
- Detailed message tracing capabilities

QA Testing Phase

The complexity and robustness of H.323 requires that the QA team define and operate heavy testing procedures. Using automated testing tools and simulating operational environments for interoperability and load testing is a must. Interoperability is required for both signaling and media streams of various codecs from different vendors to ensure the best field interoperability of the developed product. Since IP conferences run over IP networks, the simulated environment should also be protocol-aware and compliant to a given standard. An H.323 testing tool should support H.323 from version 1 up to version 4 including sub-protocols such as H.450, H.245, Q.931 and RAS. To assure comprehensive and flexible testing capabilities from the initial testing phase, it is recommended to have a testing tool with a rich library of ready-to-operate scenarios, containing modification capabilities to edit scenarios.

Interoperability

One of the major concerns for VoIP product vendor is its interoperability with other vendors, and standards-compliance. The interoperability level of the product is one of its major gradients for the product's value to the target customer. It is important to verify that the H.323 toolkit vendor participates and is active in all of the IMTC interoperability events in order to ensure better interoperability of the application. RADVISION also recommends that application developers test their products at the interoperability events as well since some of the features that were implemented on top of the toolkit can be checked with other vendors.

Load Testing

The load testing should enable a burst of messages to create traffic load and test how the developed product operates in such scenarios. It is very important to test the product, especially if it is a central device such as MCU, gatekeeper or gateway, by performing heavy load testing for at least a few days to discover the tiniest memory leaks, events and state-machine operation. The end result will be a higher quality product.

More: http://www.h323forum.org/products/#Test Equipment/Analyzers