Department of Veterans Affairs

IPv6 Enablement Support Services

IPv6 Applications Testing Best Practices

CLIN #: 0005AB

Contract: VA118-11-P-0096
Solicitation Number: VA118-11-RP-0611

June 29, 2012

Version 0.1

Revision History

Version	Purpose	Author	Function	Date
V0.1	Draft	TVG		06/29/2012

* The following symbols can be used to represent the type of change noted:

A = Added
M = Modified
D = Deleted

Table of Contents

1 EXECUTIVE SUMMARY 4

2 INTRODUCTION 5

2.1 SCOPE 7
2.2 ENVIRONMENT, ORGANIZATION AND METHODOLOGIES 8
2.3 ASSUMPTIONS AND CONSTRAINTS 9

3 GOTS APPLICATION DEVELOPMENT BEST PRACTICES (SYNOPSIS) 9

3.1 TECHNICAL IMPACT OF IPV6 ON EXISTING APPLICATIONS 9
3.1.1 Network information storage/display 9
3.1.2 Resolution and address issues 10
3.1.3 Communication APIs (raw socket code, etc.) 11
3.1.4 Socket Address Structures Specifics 12
3.2 CO-EXISTENCE METHODOLOGIES/MECHANISMS 14
3.2.1 Dual stack approaches for applications 15
3.3 INTERNET ENGINEERING TASK FORCE APPLICATION SUPPORT RECOMMENDATIONS 17
3.3.1 Application Programming Interfaces (APIs) 17

4 APPLICATION TESTING BEST PRACTICES SUPPORTING IPV6 18

4.1 TESTING PROCESS 19
4.1.1 Requirements 19
4.1.2 Software Quality Assurance 21
4.2 TEST ENVIRONMENT 21
4.2.1 USGv6 and VA Compliance 21
4.2.2 Code Base 22
4.2.3 Test Architecture 24
4.3 TEST CASES AND CRITERIA 25

List of Figures

FIGURE 1: NAME RESOLUTION 10

FIGURE 2: PROTOCOL SPECIFIC COMPARED TO PROTOCOL INDEPENDENT – IP VERSUS NAME 11

FIGURE 3: USING A COMMUNICATIONS LIBRARY TO SEPARATE APPLICATIONS FROM THE IP ADDRESS STRUCTURE 11

FIGURE 4: PROTOCOL SPECIFIC COMPARED TO PROTOCOL INDEPENDENT - STORAGE 14

FIGURE 5: IPV4 CLIENTS AND IPV4 SERVER IN AN IPV4 ONLY NODE.....16

FIGURE 6: MIXED CLIENTS AND IPV6 SERVER IN AN IPV6 ONLY NODE 17

FIGURE 7: VA IPV6 STRATEGIC OBJECTIVE.....22

FIGURE 8: TYPICAL HOST PLATFORM APPLICATION ARCHITECTURE 24

FIGURE 9: APPLICATION TEST ENVIRONMENT 24

List of Tables

TABLE 1: IPV4/ IPV6 DUAL STACK COMBINATIONS.....15

TABLE 2: DUAL STACK COMBINATIONS 16

TABLE 3: "IPV6 & USGV6 COMPLIANCE TEST CAPABILITY" REFERENCE.....20

TABLE 4: TEST CASE ONE.....25

TABLE 5: TEST CASE TWO 26

List of Appendices

APPENDIX A - REFERENCES I

APPENDIX B - GLOSSARY AND KEY TERMS.....IV

APPENDIX C - CODE EXAMPLES VII

APPENDIX D – TESTING SERVICE TESTING GUIDE XXVIII

APPENDIX E – VISTA CPRS TEXT INTEGRATION UTILITIES (TIU) GENERIC HL7 INTERFACE HANDBOOK (TEST APPENDIX) EXAMPLE FOR REVISION... XLI

1 Executive Summary

The Department of Veterans Affairs (VA) has been a leader in the United States Government's transition to the next generation Internet Protocol (IP) designated IP version 6 (IPv6) actively since 2003. As a leader, VA has continuously and consistently been at the forefront of the federal government efforts. VA's attention to detail and perseverance enabled the department to be fully prepared when the Federal CIO released a directive setting deadlines for agencies' transition to IPv6. This directive was released with the finally realistic expected exhaustion of the original IP addresses (designated IP version 4 (IPv4)) coming very shortly in the next three years. Released in September of 2010, the directive titled "Transition to IPv6" stipulated two enterprise network goals for all federal agencies, and this document is intended to support the VA's efforts in attaining the 2014 objective, and creating the technical foundation to afford the evolution of IPv6 enabled applications into the future.

The risks associated with a full enterprise transition to sustain both IPv4 inside our enterprises as well as IPv6 are staggering. One such risk is with the applications, both Commercial Off The Shelf (COTS) and Government Off The Shelf (GOTS). Realized worldwide, transitioning applications to support IPv6 is one of the greatest challenges associated with the overall enterprise transition to IPv6, in both the public and private sectors. This is due to the majority of applications today using the network in some form in a sole IPv4 environment. This is due to the long standing, appropriate for the time, practice of using the only available IP address in order to connect to other applications and databases on other platforms over the enterprise. Partly, this is due to a software design approach referred to as "Client/Server architecture", however, if for no other reason, the applications access the network to check for updates, or download for security patches. With the mission functionality provided today with the IPv4 potentially hard coded addresses embedded within the VA GOTS or mission defined software applications, the acknowledged IPv6 transition industry best practice is to commence coding all future applications to be "Protocol Independent". This Protocol Independence (PI) enables the use of either IPv4 or IPv6 as available to those applications from their platforms' respective operating systems. A date from which future software development must use the PI approach provided in this document must be established.

The last best practices element to consider is what lies beyond simple compliance. PI establishes the minimum. The focus today provided by the Federal IPv6 Task Force is on the requirement for functional parity between IPv4 and IPv6, with an emphasis, as always, on security implementations. In other words, the same functionality available within an IPv4 environment must be available in an IPv6 environment. This minimum standard is established in order to create a baseline threshold of requirements. However, industry best practices recommend that the actual designed-in benefits of applying IPv6, regarding different software design architectures, should be investigated by senior software developers and enterprise architects. Then, it is inserted into their agencies' future software architectures at the soonest opportunity. IPv6, as designed, brings multiple lessons learned from the deployment of IPv4, and so, through a thorough engineering analysis of what IPv6 enables far beyond IPv4, software developers and enterprise architects should move beyond simple IPv4 parity at their earliest opportunity.
With coding of applications comes the requisite testing. The VA enjoys a rigorous and structured software development organization, staffed with knowledgeable experts in this field. Software

testing is a known fact. However, with IPv6 involved, there are new standards and approaches to take into consideration. The development of the USGv6 Profile Process and Test Program was initiated by the OMB 05-22 memorandum to provide agencies with a common taxonomy and framework to provide acquisition specifications for IPv6 devices and an industry-based test program. The USGv6 effort was later included in the FAR as a requirement for agencies to utilize when acquiring IT devices. While the USGv6 program developed and administered by NIST goes a long way to provide agencies with a valuable tool to acquire IPv6 compliant devices, a review of specific VA mission and IT infrastructure requirements identify several areas that require expansion to meet VA's IPv6 transition needs. The VA Enablement program has chosen to establish a test program that establishes the IPv6 testing standard within the VA to meet the VA's specific mission needs. This document provides a specific testing approach in line with the VA Enablement program's initiatives regarding applications developed via an approved software engineering effort.

2 Introduction

For the past nine years, the Department of Veterans Affairs (VA) has been a leader in the United States Government's transition to the next generation Internet Protocol (IP), engaged in the discussion of transitioning to this next generation IP (designated IP version 6 (v6)), actively since 2003. As a leader, VA was involved in early initiatives led by the US Department of Defense and the Office of Management and Budget (OMB) stakeholders that drove agencies to demonstrate progress in areas of standardization and testing/certification. The culmination of this preparation for the eventual government-wide IPv6 core backbone integration and transition was the release by OMB of their memo M05-22. Based on the results, in July 2008 OMB announced that all major agencies met the June 30, 2008 deadline for successfully demonstrating their adoption of IPv6 technology.

At the time, the Department of Veterans Affairs (VA) was singled out as one of the few agencies to fully embrace the effort and achieve more than just meeting the minimum expectation. However, there was more to be accomplished. The efforts of other agencies within the government had chosen to let lapse the initiatives taken to achieve their 2008 milestone, while VA continued per their own funding capacity. Due to the lack of efforts across the federal landscape, the Federal CIO Council released the "Planning Guide/Roadmap Toward IPv6 Adoption within the US Government" in May 2009, and on page 19 offered an industry best practice transition plan outline. The document was the result of a private/public partnership in which the IPv6 industry provided a tremendous insight into the lessons learned of incorporating IPv6 into existing enterprises. This document was intended to jump-start the lagging efforts known to many to be shortly required, and VA's lessons learned were inserted liberally into this document as an example to other federal agencies.

With the finally realistic expected exhaustion of the original IP addresses (designated IP version 4 (IPv4)) coming in the next few years, the then Federal CIO, Vivek Kundra, released a memo in September of 2010 titled "Transition to IPv6" stipulating two enterprise network goals for all federal agencies:

Update public facing/external servers and services to native IPv6 by the end of FY 2012; and

Upgrade internal client applications that communicate with public Internet servers and supporting enterprise networks to use native IPv6 by end FY 2014.

The memo created the Federal IPv6 Task Force, and initiated the next stage of the entire US Government transition to IPv6.

The memo was timely. On February 3rd of 2011, the Internet Assigned Numbers Authority (IANA) distributed the last available IPv4 addresses to the five Regional Internet Registries (RIRs), signaling to the world that the approximate 4.3 billion IPv4 addresses used in the Internet and all computer networks have been issued, and no more addresses are available. On April 15th of 2011, one of the RIRs (Asia-Pacific region) exhausted its supply of IPv4 addresses, and can only provide IPv6 addresses to those who wish to create an access point to the Internet. The other four regions (including North America) are warning their respective constituents to commence transition to IPv6 now, before being forced to do so due to complete IPv4 exhaustion.

With the issuance of CIO's memo came action from the VA. The VA IPv6 Transition Project Management Office (PMO) is now established, with the appropriate governance structure. The lessons learned from the previous years' endeavors applied, and the risk management process enabled. This document is to address one of the risks identified as having the greatest impact and highest probability of occurrence, software application transition to the new IPv6 standard (while still supporting legacy IPv4 interoperability). This document is a companion to the applications development best practices due to the specific discrimination between the software developers, and those who test the applications.

Transitioning applications to support IPv6 (and accurately testing them) is one of the greatest challenges associated with the overall enterprise transition to IPv6, in both the public and private sectors. This is due to the majority of applications today using the network in some form in a sole IPv4 environment. This is due to the long standing, appropriate for the time, practice of using the only available IP address in order to connect to other applications and databases on other platforms over the enterprise. This is partly due to a software design approach referred to as "Client/Server architecture", however, if for no other reason, the applications access the network to check for updates, or download for security patches. With the mission functionality available with the IPv4 potentially hard coded addresses embedded within the VA software applications, the acknowledged IPv6 transition industry best practice is to commence coding all future applications to be "Protocol Independent". This Protocol Independence (PI) enables the use of either IPv4 or IPv6 as available to those applications from their platforms' respective operating systems. A date must be established from which all future software development must use the PI approach provided in this document. Secondly, the question to be answered is what applications can be transitioned to operate in a protocol independent manner in the most expeditious manner, which ones must wait for sufficient engineering effort, and what applications cannot be reasonably transitioned (and ultimately replaced at a much later date).

The focus provided by the Federal IPv6 Task Force is on the requirement for functional parity between IPv4 and IPv6, with an emphasis, as always, on security implementations. In other words, the same functionality available within an IPv4 environment must, at a minimum, be available in an IPv6 environment. This minimum standard is established in order to create a

baseline threshold of requirements. However, industry best practices recommend that the actual designed-in benefits of applying IPv6, regarding different software design architectures, should be investigated by senior software developers and enterprise architects. IPv6, as designed, brings multiple lessons learned from the deployment of IPv4, and so through a thorough analysis of what IPv6 enables far beyond IPv4, software developers should move beyond simple IPv4 parity at their earliest opportunity.

Developers of new applications must recognize the reality of the transition period and should build software that is tolerant and well-behaved in a heterogeneous IPv4/IPv6 environment. Although exploiting IPv6 features is an important catalyst for driving the move to IPv6, failing to support the entire community during the transition period with incompatible IPv4 solutions shall be avoided (hence the initial approach to produce PI code).

The VA IPv6 Applications Testing Best Practices document provides VA software development teams with the best practices and associative tools to support the applications' transition from IPv4 to IPv6. This best practices document, as a companion piece to the document titled, "IPv6 Applications Best Practices" dated February 28th, 2012, includes:

- IPv6 Application Development Best Practices Synopsis: A condensed guide derived from the companion document intended for VA software quality assurance and software testers to be utilized when planning testing for upgrades/modifications to existing applications or new systems to ensure IPv6 capable applications are effectively tested for acceptance and deployment.

- IPv6 Application Testing Best Practices: Specific methods and tools that are available to VA to ensure applications are tested for IPv6 operations.

2.1 Scope

A best practice is a technique or methodology that, through experience and research, has proven to reliably lead to a desired result. A commitment to using the best practices in any field is a commitment to using all the knowledge and technology at one's disposal to ensure success. The term is used frequently in the fields of health care, government administration, the education system, project management, hardware and software product development, and elsewhere.

In software development, a best practice is a well-defined method that contributes to a successful step in product development. Throughout the software industry, several best practices are widely followed. Some of the more commonly used are: an iterative development process, software quality assurance, requirement management, and change control.

A best practice tends to spread throughout a field or industry after a success has been demonstrated. However, it is often noted that demonstrated best practices can be slow to spread, even within an organization. According to the American Productivity & Quality Center, the three main barriers to adoption of a best practice are a lack of knowledge about current best practices, a lack of motivation to make changes involved in their adoption, and a lack of knowledge and skills required to do so.

2.2 Environment, Organization and Methodologies

The Department of Veterans Affairs' Systems Development Life Cycle (SDLC) is the overall process of developing, implementing, and retiring information systems through a multistep process from concept definition, concept development, system design and prototype, system development and testing, system deployment, and system operation and disposal.

The SDLC is a collaborative process across many disciplines and VA organizational entities including the Office of Enterprise Systems Engineering (ESE), Field Operations (FO), National Data Center Program (NDCP), Austin Information Technology Center (AITC), Product Development, Office of Information Protection and Risk Management (IPRM) and Field Security Operations, Veterans Health Administration (VHA), Veterans Benefits Administration (VBA), and National Cemetery Administration (NCA).

The SDLC may follow an approved model, based on the original decisions of key stakeholders and governing criteria based on size, complexity, and mission criticality of the resulting information system. Examples of such models are Waterfall, Spiral, Iterative, and Agile. Within each of these models, the following functional activities occur, with corresponding organization responsibilities.

Deployment Design is the starting phase of the overall solution lifecycle where architectural design and implementation specifications are developed and tested. The preparation of plans and specifications necessary to implement the solution are part of the deployment design stage. Developed solutions are implementation-ready for the production environment at the end of the deployment design stage.

ESE and Product Development (as applicable based on primary project ownership) are responsible for deployment design and complete release package specifications, including user and technical documentation, procurement, shipping and training plans. Architectural design artifacts, release package specifics and testing plans and results are vetted with FO and NDCP. Implementation equates to the installation and activation of newly designed or changed hardware, software, process, etc. into the production environment. This must be planned, scoped and resourced. During the implementation phase of the solution life cycle the organizations responsible work from specifications and plans created during deployment design for deploying the solution into the production environment.

FO, NDCP, and/or AITC are responsible for installation and the installation schedule, selection and validation of shipping locations/schedule, site readiness to support installation, acceptance of equipment using Asset Inventory processes, coordination and actual training working with ESE and Product Development (if applicable), communication and vetting changes through the field and FO management. ESE is responsible for Release Management, including certification of production readiness. ESE must validate that the project owner has provided an adequate release package. ESE also validates the release baseline. The 'release package' is the subset of documents, as defined by the OIT Testing and Release Checklist provided to the field along with the product. The complete Release Package will be made available to FO at deployment The 'release baseline' is defined as the product build (software and hardware specifications) along with the body of documents that support testing, install, operations, training and support of the

product. Projects will determine early in the development cycle the project's 'release baseline'. The 'release baseline' components enter change control once the project is approved for deployment.

Within this methodology and organizational responsibilities, policies and procedures must be drafted, and training must occur on these policies and procedures, which facilitate the organizational community base of enterprise architects, systems engineers, security engineers, software developers, quality assurance personnel, test personnel and deployment personnel (including service desk and security operations center personnel).

2.3 Assumptions and Constraints

The assumption of this best practices document is that the software development practices within the VA are mature, and the personnel engaged in the SDLC are appropriately trained in the practices. This document is intended for trained and experienced personnel from the VA community to apply within their respective roles and responsibilities. This includes, but is not limited to, the security engineering practices required to produce the applications in accordance with the VA's criteria. IPv6 Security best practices are outside the scope of this best practice document.

Constraints include, but are not limited to the IPv6 criteria set by the Internet Engineering Task Force (IETF), the Federal government standards, and the VA policies.

3 GOTS Application Development Best Practices (Synopsis)

This is a guide for VA software quality assurance and test teams that can be utilized when conducting SQA and developing the test plans for new developments or upgrades/modifications to existing applications or systems to ensure IPv6 is effectively implemented in the applications as recommended by the companion document.

3.1 Technical impact of IPv6 on existing applications

The largest technical impact of IPv6 on existing applications results from the expansion of address space to 128 bits. This expansion will require updates to applications that currently display, store and access (communicate) addresses in "hard coded – raw binary" format rather than symbolic/logical format. Applications will also have to incorporate approaches for resolving addresses in a dual (IPv4/IPv6) network environment.

At a generic level (as defined in "Programming Guidelines on Transition to IPv6", Tomas P. de Miguel and Eva M. Castro, January 2003) the categories of technical impact can be defined as:

- ✓ Network information storage/display
- ✓ Resolution and address issues
- ✓ Communication APIs (raw socket code, etc.)

3.1.1 Network information storage/display

Network information storage is concerned with internal representations of network addresses and the inability of existing IPv4 applications to properly handle the 128 bit addressing required for

IPv6. This includes areas such as database fields where IP addresses are used/stored, internal memory structures (i.e., device drivers), and in general wherever raw IP addresses are currently managed. Applications that configure and manage devices where IP addresses are directly used will be impacted. Collaborative or restricted access applications that register remote nodes by storing IP addresses as registry keys, and using the IP address as an identification discriminator for allowing connections from remote nodes to a session or service, will also be impacted. The ability to properly represent network information will impact applications that display such information. Programming upgrades are needed in the data storage fields, data displays, and menu options used for controlling IP addresses (e.g., Network Management Software) or displaying IP addresses as opposed to logical names.

3.1.2 Resolution and address issues

Properly managing the Domain Naming System (DNS) is a critical element of the overall IPv6 transition. Existing IPv4 applications requesting network address resolution expect to receive IPv4 addresses. Assuming the name resolver in the operating system is IPv6 capable and can accept and order DNS records from a name server that contains IPv6 addresses, the typical system call used by an IPv4 application will not work properly and will return an error. Therefore, the ability to properly resolve host names and reverse-lookups in the heterogeneous environment of mixed IPv4/IPv6 networks and nodes is key to continuity during the transition. However, as applications are upgraded, the greater use of modern, structured, programming methodologies will alleviate the impact of future modifications to standards for IP addressing and naming services. Figure 1 is an example of how applications resolve host names.

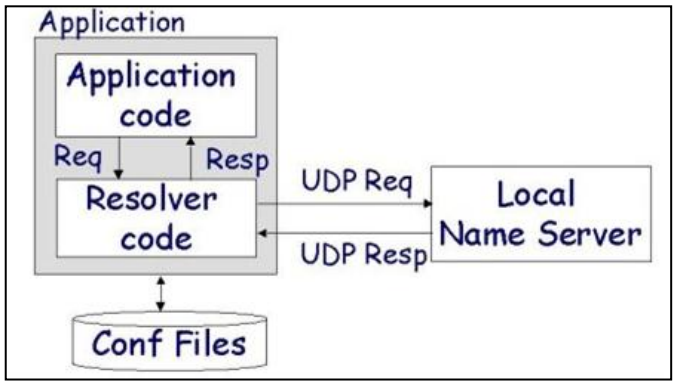

Figure 1: Name Resolution

From the best practices perspective, applications should use names instead of addresses for hosts. Names are easier to remember and remain the same, but numeric addresses could change more frequently. From an application's point of view the name resolution is a system-independent process. Applications call functions in a system library known as the resolver, typically gethostbyname and gethostbyaddr, which are linked into the application when the application is built, see Figure 1.

The resolver code is the burden of making the resolution dependent of the system configuration. Significantly appropriate, there are two new functions to make name and address conversions

protocol independent, getaddrinfo and getnameinfo. Besides, the use of these new ones instead of gethostbyname and gethostbyaddr is recommended because the latter are not normally reentrant and could provoke problems in threaded applications. The getaddrinfo function returns a linked list of addrinfo structures which contains information requested for a specific set of hostname, service and additional information stored in an addrinfo structure. The figure below gives an accurate comparison. Some additional notes are that getaddrinfo dynamically allocates memory unlike its predecessor functions, and the corresponding freeaddrinfo() function is called to free the buffers that are allocated by getaddrinfo().

IPv4 only or Protocol specific	Protocol Independent
gethostbyname() getservbyname() inet_aton() inet_pton()	getaddrinfo()
gethostbyaddr() getservbyport() inet_ntoa() inet_ntop()	getnameinfo()

Figure 2: Protocol Specific compared to Protocol Independent – IP versus Name

3.1.3 Communication APIs (raw socket code, etc.)

Any application that currently communicates with raw sockets will require modification. It is anticipated that many older communications libraries will be significantly impacted. Communication APIs will need to be individually assessed. For some older applications, there could be considerable work in just updating the compilers and development environment in order to utilize the newer calls. In upgrading to IPv6, applications need to use more generic calls, which support both IPv4 and IPv6 information. Figure 3 below illustrates a simplified architecture situation where the application ideally remains independent of IPv4/IPv6 address structure differences through the use of a communications library.

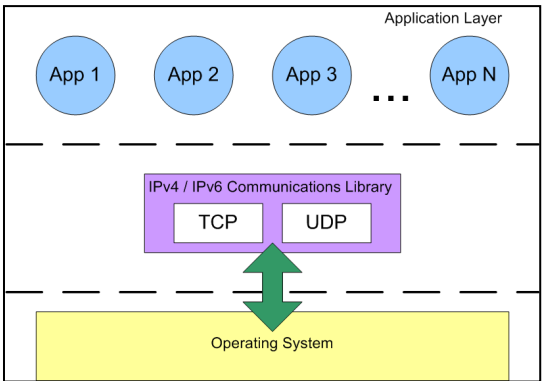

Figure 3: Using a Communications Library to Separate Applications from the IP Address Structure

Older software libraries should be considered carefully during the assessment process in order to decide whether a co-existence approach is more appropriate than the level of resource investment that would be required to upgrade the code base.

3.1.4 Socket Address Structures Specifics

Functions provided by socket API use the socket address structures to determine the communication service access point. Since different protocols can handle socket functions, a generic socket address structure is used as an argument of these functions for any of the supported communication protocol families, sockaddr.

```
Struct sockaddr {
     sa_family_t sa_family;        /* Address family */
     char sa_data[14];             /* protocol-specific address
*/
};
```

Although socket functions handle the generic socket address structures, developers must fill the adequate socket address structure according to the communication protocol they are using to establish the socket. In particular, the IPv4 sockets use the following structure, sockaddr_in:

```
typedef uint32_t in_addr_t;
struct in_addr  {
   in_addr_t s_addr;              /* IPv4 address */
};

struct sockaddr_in {
    sa_family_t sin_family;          /* AF_INET */
    in_port_t sin_port;             /* Port number.  */
    struct in_addr sin_addr;         /* Internet address.  */

    /* Pad to size of `struct sockaddr'.  */
    unsigned char sin_zero[sizeof (struct sockaddr) –
                    sizeof (sa_family_t) –
                    sizeof (in_port_t) –
                    sizeof (struct in_addr)];
};
```

And the IPv6 sockets use the following structure, sockaddr_in6, with a new address family AF_INET6:

```
struct in6_addr {
   union {
      uint8_t u6_addr8[16];
      uint16_t u6_addr16[8];
      uint32_t u6_addr32[4];
   } in6_u;
```

```
   #define s6_addr              in6_u.u6_addr8
   #define s6_addr16             in6_u.u6_addr16
   #define s6_addr32             in6_u.u6_addr32
};

struct sockaddr_in6 {
   sa_family_t sin6_family;    /* AF_INET6 */
   in_port_t sin6_port;        /* Transport layer port # */
   uint32_t sin6_flowinfo;     /* IPv6 flow information */
   struct in6_addr sin6_addr;  /* IPv6 address */
   uint32_t sin6_scope_id;     /* IPv6 scope-id */
};
```

The sockaddr_in or sockaddr_in6 structures are utilized when using IPv4 or IPv6, respectively. Existing applications are written assuming IPv4, using the sockaddr_in structure. They can be easily ported changing this structure by sockaddr_in6.

However, when writing portable code, it is preferable to eliminate protocol version dependencies from source code. There is a new data structure, sockaddr_storage, large enough to store all supported protocol-specific address structures and adequately aligned to be cast to the a specific address structure.

```
/* Structure large enough to hold any socket address (with the historical exception of
AF_UNIX). 128 bytes reserved.  */

#if ULONG_MAX > 0xffffffff
# define __ss_aligntype __uint64_t
#else
# define __ss_aligntype __uint32_t
#endif
#define _SS_SIZE        128
#define _SS_PADSIZE     (_SS_SIZE – (2 * sizeof (__ss_aligntype)))

struct sockaddr_storage
{
   sa_family_t ss_family;      /* Address family */
   __ss_aligntype __ss_align;  /* Force desired alignment.  */
   char __ss_padding[_SS_PADSIZE];
};
```

The use of sockaddr_in or sockaddr_in6 structures along with the new data structure, sockaddr_storage, will allow software developers to specify the required IP address. However, this is to be used as a transition mechanism to ultimately support the use of IPv6 benefits over the legacy IPv4 employment. Software Requirements Specifications written to establish sole IPv6 connectivity must make the above changes, but must be approved given the minimum

preferred approach to employ PI practices and make calls for FQDNs to establish host to host connections. The figure below provides a side by side representation for the storage of IP addresses, should they be absolutely required.

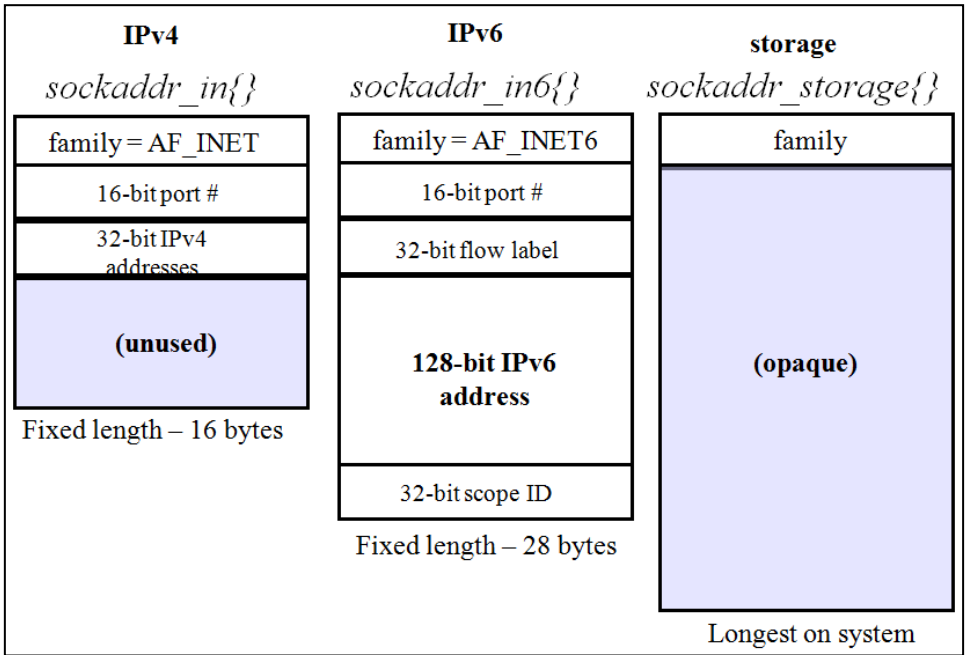

Figure 4: Protocol Specific compared to Protocol Independent - Storage

3.2 Co-existence methodologies/mechanisms

An unidentified (yet assumed with a firm certainty) set of legacy applications will not or should not necessarily be upgraded to native IPv6. This is a common risk element identified in every industry best practice risk register. In these cases, it is important to understand that co-existence approaches exist that will still allow these applications interoperability and compatibility with IPv6 networks and systems, but are not mandated. The idea behind co-existence is that the transition from IPv4 to IPv6 will take time, likely several years. The IPv6 industry agrees that some legacy systems will remain IPv4 indefinitely due to mission criticality, code base reverse engineering complexities, and legacy platform technical incompatibilities, to name the primary rationales for not transitioning.

Co-existence mechanisms can be based on network or software approaches, or a combination of both. This is an area which is a paradigm shift for the future of software design and application development, and requires education at various levels of engineering skill sets. The co-existence mechanism we will be testing to in this document, is the Dual stack approaches (network and software system). Use of translation plug-ins / software proxies as a transition approach is not within the scope of this document due to the severely increased risks in performance and interoperability by their use.

Each of the co-existence mechanisms imposes system constraints and limitations, which must be assessed in the context of the overall proposed deployment/transition approach. Some of these mechanisms will be driven by the applications themselves as well as the time frame for

deploying capabilities into operations. The industry best practice is to commence designing and writing protocol independent code to support a dual stack environment at the earliest opportunity, and then during tech refresh cycles in the software development life cycle evaluate the opportunity to shift the legacy code to a protocol independent version.

3.2.1 Dual stack approaches for applications

For network items such as routers, the dual stack approach entails (at a very simplified level) turning on a parallel capability which can deal with either IPv4 or IPv6 packets or a combination of the two. At a high abstraction level, the dual stack model can also be applied to software applications. However, there are some details that must be understood.

Note that dual stack capability for applications should not be confused with dual stack availability in the operating system. Even when an application is dual stack enabled, it will still require IPv6 support from the underlying operating system in order to interact with other devices and applications on the network.

The table below illustrates a subset of the IPv4/IPv6 dual stack combinations. It describes servers and client components that are either dual stack enabled (i.e., understand and communicate both IPv4 and IPv6 protocols) or support only a single protocol. The table also includes certain configurations where communication is not possible (N/P).

The highlighted column in the following table and Figure 5 below illustrate an IPv4 server in an IPv4 only application node. If both IPv4 and IPv6 network interfaces are available but there are only IPv4 applications, IPv6 will not be used.

Table 1: IPv4/ IPv6 Dual Stack Combinations

IPv4/IPv6 Dual Stack Combinations					
			IPv4 Server	IPv6 Server	
		IPv4 Only	Dual Stack	IPv6 Only	Dual Stack
IPv4 Client	IPv4 Node	IPv4	IPv4	N/P	IPv4
	Dual Stack	IPv4	IPv4	N/P	IPv4
IPv6 Client	IPv6 Node	N/P	N/P	IPv6	IPv6
	Dual Stack	IPv4	IPv4	IPv6	IPv6

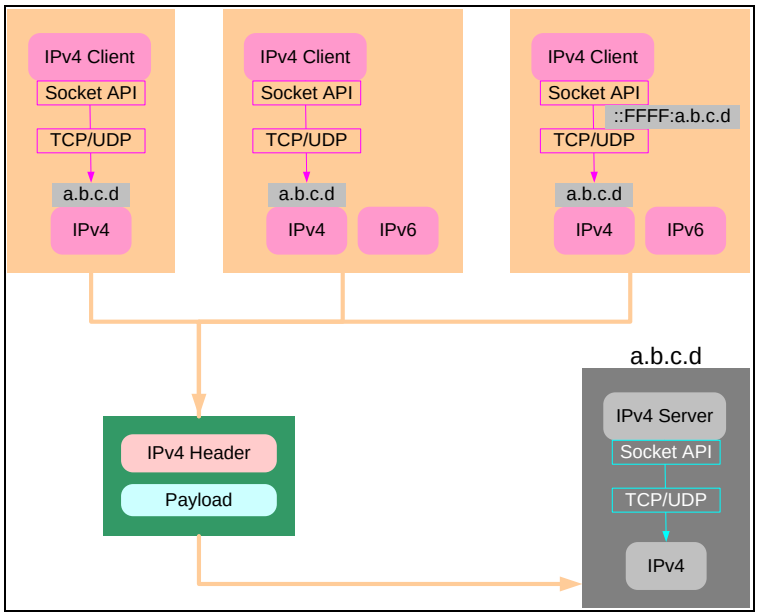

Figure 5: IPv4 Clients and IPv4 Server in an IPv4 Only Node

Again, note that dual stack capability for applications should not be confused with dual stack availability in the operating system.

The highlighted column in the following table and Figure 6 below illustrate an IPv6 server in an IPv6 only application node.

Table 2: Dual Stack Combinations

IPv4/IPv6 Dual Stack Combinations					
		IPv4 Server		IPv6 Server	
		IPv4 Only	Dual Stack	IPv6 Only	Dual Stack
IPv4 Client	IPv4 Node	IPv4	IPv4	N/P	IPv4
	Dual Stack	IPv4	IPv4	N/P	IPv4
IPv6 Client	IPv6 Node	N/P	N/P	IPv6	IPv6
	Dual Stack	IPv4	IPv4	IPv6	IPv6

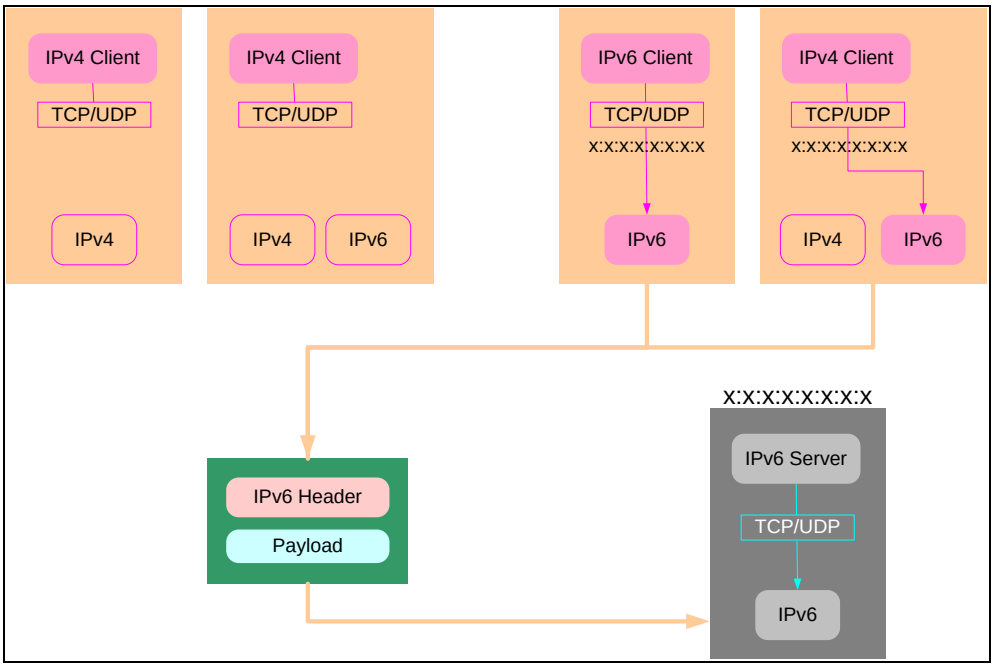

Figure 6: Mixed Clients and IPv6 Server in an IPv6 Only Node

The application transport module is directly related to the communication establishment and the information transmission. However, this application could have dependencies on other modules, which may have their own IP address dependencies, so they are IP version dependent. As stated previously, some applications require an IP address as an argument to establish a new connection to this address. As an example using the client/server model, the client needs to know the IP address or the hostname where the server is running.

The use of Fully Qualified Domain Name (FQDN) instead of IP address is preferable since nodes can change their addresses and this process should be transparent to applications. Applications can store and use FQDN, delegating the resolution of the IP addresses to the name resolution system, which will return the associated IP address at the moment of the query.

3.3 Internet Engineering Task Force (IETF) Application Support Recommendations

As described in RFC 6434-"IPv6 Node Requirements", IPv6 will be deployed in a wide range of devices and situations. Specifying the requirements for IPv6 nodes allows IPv6 to function well and interoperate in a large number of situations and deployments. Applications play a tremendous role in defining the network topology, since technology is intended to enhance mission effectiveness and productivity. The following recommendations have been included in this best practice document to certain degrees of fidelity, and each quoted RFC forms the final basis for any future Software Requirements Specification (SRS). This includes Application Programmer Interfaces (APIs).

3.3.1 Application Programming Interfaces (APIs)

There are a number of IPv6-related APIs. This document does not mandate the use of any, because the choice of API does not directly relate to on-the-wire behavior of protocols. Implementers, however, would be advised to consider providing a common API or reviewing existing APIs for the type of functionality they provide to applications.

"Basic Socket Interface Extensions for IPv6" [RFC3493] provides IPv6 functionality used by typical applications. Implementers should note that RFC3493 has been picked up and further standardized by the Portable Operating System Interface (POSIX) [POSIX].

The above API is the focus for the foundational code to produce a protocol independent application code base. Once application developers more fully understand IPv6, and enhanced requirements can be delineated, the below APIs would be included (but are out of scope of this document).

"Advanced Sockets Application Program Interface (API) for IPv6" [RFC3542] provides access to advanced IPv6 features needed by diagnostic and other more specialized applications.

"IPv6 Socket API for Source Address Selection" [RFC5014] provides facilities that allow an application to override the default Source Address Selection rules of [RFC3484].

"Socket Interface Extensions for Multicast Source Filters" [RFC3678] provides support for expressing source filters on multicast group memberships.

"Extension to Sockets API for Mobile IPv6" [RFC4584] provides application support for accessing and enabling Mobile IPv6 [RFC6275] features.

4 Application Testing Best Practices supporting IPv6

Software testing is done every day within the Software Development Life Cycle (SDLC) adopted by the respective organization. The periodicity of the iterations of code, debug, test, deploy is dependent on that SDLC, whether it be waterfall or agile, and determined far in advance of the establishment of the requirements. The purpose of this document is to not "re-invent the wheel" and depict how this testing is applicable to this or that SDLC. Rather, this document is intended to establish a testing paradigm that fits well within any SDLC chosen, based on the coding best practices detailed in the companion document[1].

All applications to be developed, being developed or currently deployed (i.e., still operational, not retired) within the VA enterprise must be considered as candidates for establishing or refining the code in order for the code to support the stated PI requirements. Although this is stated in the companion document "IPv6 Applications Best Practices", this is worthy of re-iteration here. Such applications as stated above must be identified, categorized and prioritized within the VA engineering and programmatic administration. Categories are as follows:

- A new system that has not completed the requirements gate review

[1] "IPv6 Applications Best Practices" dated February 28th, 2012; Deliverable 0005AA

- A new system, developed and under test, that has not yet been deployed
- A system already deployed that can be upgraded to IPv6
- A system that should be upgraded to IPv6
- A system considered to be legacy and therefore, has no plans for upgrading to IPv6

For the sake of scoping, this document is intended to test any application that has gone through the process of integrating the PI standard, regardless of its lifecycle stage (e.g. new or revised).

4.1 Testing Process

The process outlined below is a "best practice" regarding the insertion of the PI standards into the SDLC. This is not intended to suggest alteration to the existing test strategy outlined in Appendix D (VA Testing Service Testing Guide) nor make changes to the VA Software Quality Assurance/Test process. Based on review of both VA documents, the essential components are offered here which should be inserted into the already in place and mature process. The current tabs contained in the VA SQA/Test Review Checklist can be easily modified to incorporate the PI standard. Appendix E provides an example of testing documentation for a VistA module that would require modification should that module be selected to be refreshed to reflect the PI standard.

At the least, the following is suggested:

- ✓ Confirm Software Requirements Specification
  - Defined PI coding standards
  - Establish platform and network operating environment IPv6 standards
- ✓ Confirm test environment supports development and production environment
- ✓ Establish and Execute the Test and Evaluation Master Plan (which elements contain)
  - Provide SQA personnel with accurate and tested code examples to compare between them and the application under test
  - Create Acceptance Criteria (SQA and Functional Test) traceable to requirements
  - SQA conduct code review (Pass/Fail criteria)
  - Create Test Case(s) traceable to acceptance criteria
  - Configure test environment per test case
  - Schedule and conduct tests
  - Review results (Pass/Fail)

4.1.1 Requirements

In any SDLC, the requirements component is essential. A System Requirements Review (SRR) is a normal gate with which the process ensures that threshold and objective requirements, both functional and standards compliance based, must be documented and established for traceability. Artifacts associated with a SRR include:

- ✓ Requirements Traceability Matrix (RTM)
- ✓ List of discrepancies/questions to address at the SRR
- ✓ Draft Functional Baseline
- ✓ Draft System Specification and any initial draft Item Performance Specifications
- ✓ Trade studies
- ✓ Risk assessments

This document's purpose is to provide the best practices regarding testing a PI application so that the application under test can be verified that it has incorporated the software coding standard provided in the companion document, and as mandated to be included as a technical standard within the system's System Requirement Specification (SRS). The following provides the initial RTM mapping to these standards.

The IPv6 Enablement document titled "IPv6 & USGv6 Compliance Test Capability", dated February 23, 2012 establishes applications as the 6th element to be created as an enterprise entity with associated standards and subsequent testing requirements. The rationale for this specificity is that there are legacy applications existing within the VA enterprise today that cannot support the IPv6 environment.

Table 3: "IPv6 & USGv6 Compliance Test Capability" Reference

9 VA IPv6 Profile Development Classes 9.5 Applications / SW Not covered within the USGv6 definitions, the VA must also require that software being procured conform to IPv6-specific requirements. In the most general sense, all software should be capable of functioning in both Dual-Stack and IPv6-only modes of operation. Important considerations for IPv6-enabled software are: • The software application can function the exact same way in an IPv6-only environment, and • The software coding for all network interfaces is in compliance with numerous IEEE and IETF standards supporting IPv6 socket implementation.

The broad requirements called out from section 9.5 above lack the specificity needed to sufficiently create the test cases and criteria. The VA employs a standard template for a Software Requirements Specification (SRS), from which all software testers and SQA base their process actions. The code specifications may be further refined as follows.

When porting to IPv6, most of changes will be made in the application transport module, which establishes communications to remote nodes. The developer, either in creating a new application, or revising a legacy IPv4 specific application, must separate the transport module from the rest of application functional modules. This separation makes the application independent on the network system used. Then, if the network protocol is changed, only the transport module should be modified. The transport module should provide the communication channel abstraction with basic channel operations and generic data structures to represent the addresses. These abstractions could be instantiated as different implementations depending on the network protocol available at the time required. The application should deal with this generic communication channel interface without knowing the network protocol used (and with the current advent of operating systems supporting dual stack operations, the default is the establishment of IPv6 first, followed, if IPv6 is not available, per the OS' timing standards to the establishment of the IPv4 connection. The crux of the design implementation is allowing the OS

to perform per design, and then afford the application to make the required connections to another host. Using this design, the network protocol employed is irrelevant to the application.

Once the transport module has been designed, there are some implementation details related to the type of the nodes which will run the application: IPv4-only nodes, IPv6-only nodes or both, dual stack. We are focused on developing (and testing) applications that may operate effectively on any of these installations, since the application will establish host and socket connectivity by employing Fully Qualified Domain Names (FQDN) contained within the Domain Name Server (DNS), and therefore avoid the requirement to establish network specific connections.

Section 3 and Appendix C of this document provides the specific details to be assessed by SQA during their source code reviews, and by the testing team in developing the test cases to verify the functional capability of the application once executed.

4.1.2 Software Quality Assurance

SQA reviews the source code once the developer has completed the assigned effort and has successfully compiled the code into an executable (in this case). The source code, in accordance with best practices, is supposed to be documented in a similar fashion as the example in Appendix C, although one could state the code in Appendix C is documented more than normally with specific information regarding the PI elements in order to support code and document comparison.

SQA relies on their knowledge and experience, the SRS, the VA policies, and, when available, validated code examples, in order to ensure the developer has produced the right application, the right way. Due to the varieties of SDLC, the Software Lines of Code (SLOC) required to create functions and calls can be constrained by the design, and the timing. However, due to the relatively low SLOC, and the straightforward one for one transition of code, this SQA effort may only take of few moments of time, given the size of the transport module. Once approved, the executable is transitioned and installed in the test environment.

4.2 Test Environment

The test environment is critical to the understanding of the coded application's behavior once the developer has successfully coded the application to be protocol-independent.

4.2.1 USGv6 and VA Compliance

The test environment platform, in order to support IPv6 testing, must meet the criteria established by the VA for, at least, production devices employed in supporting the business functions supported by the corresponding application. The minimum criteria regarding RFC compliance for the platforms engaged in establishment of the application test environment may be found in section 6.11 of NIST SP 500-267 Vol. 1 (Pages 35-37). The essence of the testing is to confirm that the developers successfully compiled code in accordance with the IPv6 Enablement document titled "IPv6 Applications Best Practices" dated February 28th, 2012. In order to test this code to meet minimum acceptable criteria, the platforms must be compliant with the underlying RFCs.

The IPv6 Enablement document titled "IPv6 & USGv6 Compliance Test Capability" dated February 23rd, 2012 in section 6 details the test classes within which this test environment construct must perform. The class to be used for this testing is "functional", depicted in that document's figure 16. Section 6.1.7 of that document refers, specifically addressing application testing, however lacks the specificity that this document provides. In section 7 of same document, the definition of the test environment is specified. The application test environment to be established per this document is based on objectives 3 and 4, in which PI code is best suited to optimize the transition to a homogeneous native IPv6 environment. The VA's strategic intentions is depicted in Figure 7.

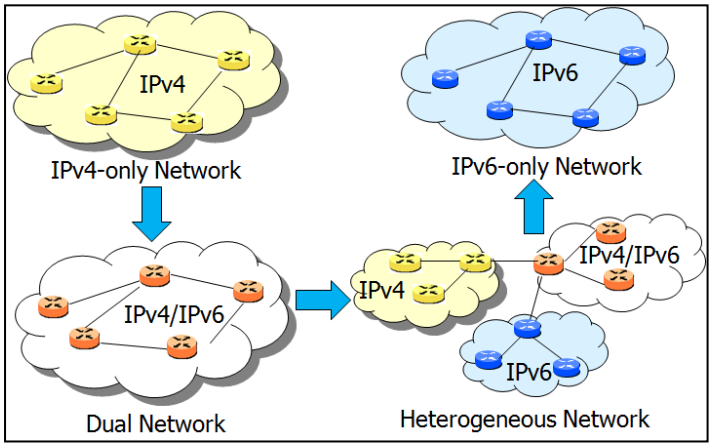

Figure 7: VA IPv6 Strategic Objective

4.2.2 Code Base

The knowledge of the code base, since this is the foundation of the application under test, is a pre-requisite for test planning. Given the nuances of the different bases, in order to establish the credibility of the code with respect to IPv6 and corresponding APIs, the SQA and test planners need to understand the circumstances with the potential for legacy applications to effectively create appropriate test cases. This will be realized fully during the revision of legacy applications employing various code bases. However, in the requirements definition for new applications, the versatility of the code base regarding IPv6 must be taken into consideration at the time of requirements review.

The IETF has created several RFCs to give guidance on C-language IPv6 socket programming which are presented in section 3.3.2 and a complete documented example in Appendix C. However, the following is provided as precise examples of what is available today with the existing popular code bases.

JAVA

Java has supported IPv6 for many years. IPv6 support was added in J2SDK/JRE 1.4. Java supports address types Inet4Address and Inet6Address and address-family independent sockets.

Significantly, for developers, with Java, the IPv6 stack is preferred by default. Unfortunately, at this time Java prefers IPv4 addresses returned by DNS.

The following two settings control how Java uses either IP version:
Java.net.preferIPv4Stack= < true|false > # false by default
java.net.preferIPv6Addresses= < true|false > # false by default
Therefore, these lines should be added the java options.
-Djava.net.preferIPv4Stack=false
-Djava.net.preferIPv6Addresses=true

PYTHON
Python has good IPv6 support in Python versions 2.3 and 2.6, and the version 2.7.1 has a few bug fixes regarding IPv6. Python has many protocol independent functions such as: socket.AF_INET, socket.AF_INET6, socket.getaddrinfo, socket.getnameinfo, socket.socket([family[, type[, proto]]]), and socket.inet_pton, socket.inet_ntop. There is a simple way to check the version of Python for IPv6 capabilities.

Enter the following lines:
# python
>>> import socket
>>> socket.has_ipv6
True

.NET
Microsoft has supported IPv6 for over 10 years. They are one of the leaders as a vendor in providing APIs for protocol independent network applications.

Examples of what they have instituted:
System.Net, RPC, wininet, sockets, DNS, HTTP, XML Web Services
Additionally, Developer PSS support is available. Most managed applications within the Microsoft inventory (including ASP.NET, XML Web services) have been designed to work over IPv6. Enable IPv6 on the platform with <netsh int ipv6 install> to confirm. Server version 2003 and later, and client XP and later have increasing mature IPv6 capability. The optimum platforms today to port code to would be Server 2008 (Release2) and the Win7 client. An example of the representative PI code is contained in Appendix C. SQA personnel may use this as a benchmark when conducting a code review (since the code is validated by Microsoft, and used to test the efficacy of their IPv6 code checker Checkv4.exe, available in MSDN).

PERL
Perl CORE has no IPv6 support. Developers must use either Socket6 and IO::Socket::INET6 or Socket and IO::Socket::INET. No single API will work for IPv4-only, dual-protocol, and IPv6-only and this makes it difficult to produce PI code. Therefore, developers must re-write the code to establish IPv6 connectivity, let alone PI compliance.

Determine if the current PERL version has IPv6 support using these following commands:
$ perl –Msocket6 –e1
$ perl –MIO::Socket::INET6 –e1

Since this is a best practice document, the respective code bases above are provided as considerations for the SQA and test team, yet are out of scope for this test document regarding these specific code bases for testing. The overarching capability, with the most documented code base (IETF references pertain), is C. Experienced developers, and experienced SQA and software test engineers, can easily extrapolate the calls and threads based on these guidelines employing the respective RFCs and C code examples.

4.2.3 Test Architecture

Figure 8 provides a representation of the application sequence in a dual stack environment.

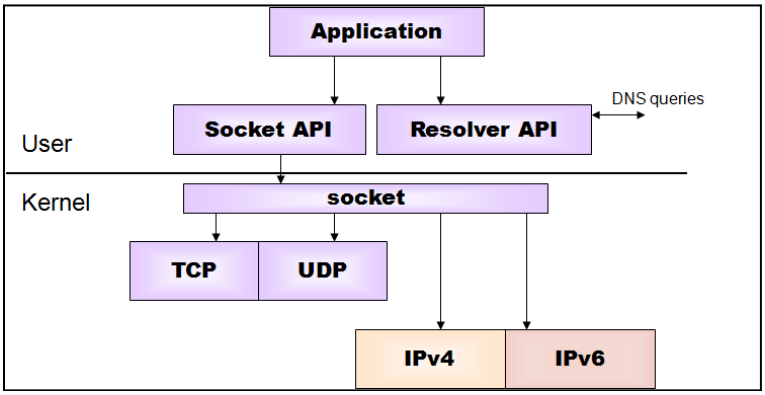

Figure 8: Typical Host Platform Application Architecture

Given that the primary issue involved in creating the test architecture is to establish an emulation of the production environment, we must at least establish two hosts both containing the application under test between which they may establish communications across the network. Given that the test environment also must be employing IPv4 and IPv6, and that since the PI standard requires the use of the DNS, the architecture also requires DHCP/DHCPv6 and DNS (capable of A and AAAA records). Figure 9 provides an example of the architecture.

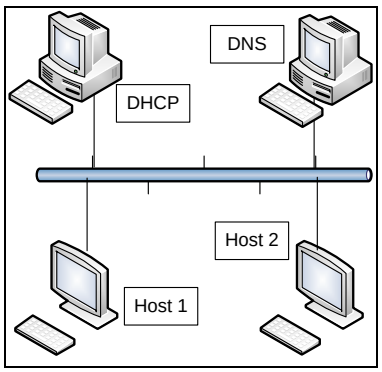

Figure 9: Application Test Environment

The test environment is configured to provide stateful IPv4 and IPv6 addresses assigned via the DHCP, and the DNS is configured with the appropriate VA DNS naming conventions associated for the corresponding records and IP addresses. The addresses available must be in accordance

with the VA IPv6 Addressing Test Allocation policy provided within the IPv6 Addressing plan. As stated earlier, all components of the test environment must be compliant with the VA IPv6 Compliance policy. Since the platform is for testing the application, the recommendation is to establish the entire platform employing Microsoft products for homogeneity and thus reduction of risk. Wireshark is operating at the time of test on one device on network for PCAP and test point analysis.

Prior to start of each test case, the test devices are powered up and operating properly with all syslogs and appropriate functions operating per established standards, appropriate network connectivity is established per RFC criteria, VA standards, and test case requirements, and the PCAP application is verified to be working in accordance with vendor specs, VA standards, and configured to support the test case per requirements.

4.3 Test Cases and Criteria

The test acceptance criteria and thus the test case must be built from the requirements. The functional acceptance criteria is that the application operates in either an IPv4 or IPv6 network environment, employing FQDNs without relying on specific IP addresses. Given that in a dual stack configuration, the host OSs choose IPv6 first, and then shift to IPv4 should there be no IPv6 enabled on the network, this provides the basis for two test cases. One is to be run with both IPv4 and IPv6 enabled on all test devices, and one is to be run with only IPv4 is enabled.

Table 4: Test Case One

Test Case One		**IPv6 (in Dual Stack with IPv4 enabled)**		
Step	**Performance Steps**	**Expected Result**	**P/F/Partial/**	**Notes:**
1.	Begin packet capture on the PCAP device.	Normal operating condition	Not Test Criteria	Required for test documentation and test criteria status
2.	Open syslogs on all devices on network	Normal operating condition	Not Test Criteria	Required for test documentation and test criteria status
3.	Execute application under test	Application should commence start up in stack per runtime.		Monitor syslogs, CPU cycles and RAM usage on application host platform for abnormal operation
4.	Observe and capture PCAP traffic	Calls should ,in sequence, be to the DNS and then to the other host device		Given the test environment is properly configured
5.	Observe and capture syslog reports on both host platforms	Application communication connection between devices is established		
6.	Within the application under test, execute a command that requires data transfer between hosts.	Data transfer occurs between hosts		PCAP confirms packet traversal in timing coincident to command function execution.

7.	Exit application	Wait for full memory to be purged		Observe syslogs
8.	Document test			All logs, PCAP logs, and screen captures

Test case one is in a dual stack mode with both IPv6 and IPv4 enabled. Test case two is intended to test the applications ability to operate in an IPv4 enabled network. Due to the OS functionality of the hosts, this will also validate the applications capability to interface with the host OS due.

Test environment configuration is identical to test one's with the exception that DHCPv6 is not configures to provide IPv6 addressing, and the DNS does not have the DNS name records associated with IPv6 addresses.

Table 5: Test Case Two

Test Case Two		**IPv4 (in Dual Stack with IPv6 disabled on DHCP and DNS)**		
Step	**Performance Steps**	**Expected Result**	**P/F/Partial/**	**Notes:**
1.	Begin packet capture on the PCAP device.	Normal operating condition	Not Test Criteria	Required for test documentation and test criteria status
2.	Open syslogs on all devices on network	Normal operating condition	Not Test Criteria	Required for test documentation and test criteria status
3.	Execute application under test	Application should commence start up in stack per runtime.		Monitor syslogs, CPU cycles and RAM usage on application host platform for abnormal operation
4.	Observe and capture PCAP traffic	Calls should ,in sequence, be to the DNS and then to the other host device		Given the test environment is properly configured
5.	Observe and capture syslog reports on both host platforms	Application communication connection between devices is established		
6.	Within the application under test, execute a command that requires data transfer between hosts.	Data transfer occurs between hosts		PCAP confirms packet traversal in timing coincident to command function execution.
7.	Exit application	Wait for full memory to be purged		Observe syslogs
8.	Document test			All logs, PCAP logs, and screen captures

APPENDIX A - References

The following table summarizes the material referenced in this document.

Document Name	Originator/Version/Date	Description
Guidelines for the Secure Deployment of IPv6	NIST SP 800-119; December 2010	Due to the exhaustion of IPv4 (Internet Protocol version 4) address space, and the Office of Management and Budget (OMB)1 mandate that U.S. federal agencies begin to use the IPv6 (Internet Protocol version 6) protocol, NIST undertook the development of a guide to help educate federal agencies about the possible security risks during their initial IPv6 deployment. This document provides guidelines for organizations to aid in securely deploying IPv6.
Transition to IPv6	OMB; September 28, 2010	This memo describes specific steps for agencies to expedite the operational deployment and use of IPv6. This set the 2012 and 2014 objectives and dates.
USGv6 Test Methods: General Description and Validation	NIST SP 500-273; November 30, 2009 (Original; updated versions available on-line)	This document forms part of the USGv6 Testing Program. It is specifically directed at: Accreditation organizations who are signatory to the International Laboratory Accreditation (ILAC) mutual recognition arrangement; Testing laboratories who will apply to such an accreditor for USGv6 profile testing accreditation, and; Test method developers who develop abstract and/or executable tets and test methods for USGv6 capable hosts, routers and network protection devices. Taken together with the abstract test specifications published at the USGv6

		testing website [14] it provides the essential material for accreditors to establish testing programs compliant with ISO/IEC 17025 [3] and for test laboratories to seek accreditation for USGv6 test methods.
Planning Guide/Roadmap Toward IPv6 Adoption within the U.S. Government	Federal CIO Council; Version 1.0; May 2009	The purpose of this document is to aid in understanding the Federal Government's Internet Protocol version 6 (IPv6) vision and to provide specific guidance for adopting and exploiting this protocol for business and technology. Based on the information in this document, the chief architect or CIO should be able to develop and explain the business case for IPv6 transition, develop a Target Architecture and Transition Strategy Plan to guide the agency's IPv6 transition, and integrate IPv6 requirements into impacted federal initiatives.
A Profile for IPv6 in the U.S. Government	NIST SP 500-267; Ver 1.0; July 2008 (Original; updated versions available on-line)	This document recommends a technology acquisition profile for common IPv6 devices to be procured and deployed in operational USG IT systems. This standards profile is meant to: (a) define a simple taxonomy of common network devices; (b) define their minimal mandatory IPv6 capabilities and identify significant configuration options so as to assist agencies in the development of more specific acquisition and deployment plans; and, (c) provide the technical basis upon which future USG polices can be defined.
IPv6 Transition Guidance	Federal CIO Council Architecture and Infrastructure	This document provides the single most comprehensive federal guidance

	Committee; February 2006	regarding the establishment of training within respective agencies in order to support a transition to IPv6. Although specifically tied to the OMB 05-22 memorandum and associated timeframe, the process remains appropriate, and, as a reference, is still applicable.
Transition Planning for Internet Protocol Version 6 (IPv6)	OMB; M-05-22; August 2, 2005	This memorandum and its attachments provide guidance to the agencies to ensure an orderly and secure transition from Internet Protocol Version 4 (IPv4) to Version 6 (IPv6). Since the Internet Protocol is core to an agency's IT infrastructure, beginning in February, 2006 OMB will use the Enterprise Architecture Assessment Framework to evaluate agency IPv6 transition planning and progress, IP device inventory completeness, and impact analysis thoroughness.
IPv6 Node Requirements	RFC 6434; December 2011	This document defines requirements for IPv6 nodes. IPv6 will be deployed in a wide range of devices and situations. Specifying the requirements for IPv6 nodes allows IPv6 to function well and interoperate in a large number of situations and deployments.
"Programming Guidelines on Transition to IPv6"	Tomas P. de Miguel and Eva M. Castro, January 2003	
"IPv6 Network Programming"	Jun-ichiro itojun Hagino, ISBN: 1-55558-318-0, 2004	

APPENDIX B - ***Glossary and Key Terms***

The following table provides definitions and explanations for terms and acronyms relevant to the content presented within this document.

AAAA: Authentication, authorization, accounting and auditing
ACL: Access Control list
ARIN: American Registry for Internet Numbers
ARP: Address Resolution Protocol
CCTV: Closed circuit television
CPIC: Capital Planning and Investment Control
COI: Communities of Interest
COOP: Continuity of Operations Plan
DHCPv6: Dynamic Host Configuration Protocol for IPv6
DISR: DoD Information Standards Registry
DMZ: Demilitarized zone
DNS: Domain Name System
DNSSEC: Domain Name System (DNS) Security Extensions
DoD: Department of Defense
DOL: Department of Labor
DOT: Department of Transportation
EA: Enterprise Architecture
EAAF: Enterprise Architecture Assessment Framework
E-Authentication: Electronic Authentication
ED: Education
E-Mail: Electronic Mail
e-Gov: Electronic Government
FDCC: Federal Desktop Core Configuration
FEA PM: Federal Enterprise Architecture Program Management Office
Firewall: A firewall is a dedicated appliance, or software running on another computer, which inspects network traffic passing through it, and denies or permits passage based on a set of rules.
FTP: File Transfer Protocol
GRE: Generic Routing Encapsulation
HSPD-12: Homeland Security Presidential Directive (HSPD) 12 is "Policy for a Common Identification Standard for Federal Employees and Contractors"
IA: Information Assurance
IDS: Intrusion detection system
IETF: Internet Engineering Task Force
IKE: Internet Key Exchange
IP Fax: Internet Protocol Facsimile
IPS: Intrusion prevention system
IPsec: Internet Protocol Security

IPT: Integrated Project Team
IPv4: Internet Protocol Version 4
IPv6: Internet Protocol Version 6
IRS: Internal Revenue Service
ISATAP: Intra-Site Automatic Tunnel Addressing Protocol
IS-IS: Intermediate system to intermediate system
ISP: Internet Service Provider
IT: Information Technology
ITI LoB: Information Technology (IT) Infrastructure Line of Business
ITI PPMO: IT Infrastructure Program Performance Measurement Office
ITS: Intelligent Transportation Systems
ITV: In-Transit Visibility
LAN: Local area network
LOB: Line of Business
MAC ID: Media Access Control Identification. MAC is also known as Medium Access Control
MIPv6: Mobile IP version 6
MPLS: Multi Protocol Label Switching
MTU: Maximum Transfer Unit
NAT: Network Address Translation
NEMO: Network mobility
NETWORX: Federal Government contract vehicle for telecommunications and related services
NIST: National Institute for Standards and Technology
NMS: Network Management System
OMB: Office of Management and Budget
OS: Operating system
OSPF: Open Shortest Path First
PC's: Personal computers
PDA: Personal digital assistant
PKI: Public key infrastructure
QoS: Quality of service
RFC: Request for Comments
RFP: Request for Proposal
RIR: Regional Internet Registry
SDK: Software Development Kit
SNMP: Simple Network Management Protocol
SOA: Service-oriented architecture
SRM: Secure Remove
SSL: Secure Sockets Layer
TCP/IP: Transmission Control Protocol/Internet Protocol
TIC: Trusted Internet Connection

TRM: Time Reversal Mirror
UC: Unified Communications
USG IPv6 FAQ's: United States Government IPv6 Frequently Asked Questions
USPS: U.S. Postal Service
VLAN: virtual LAN
VoIP: Voice over Internet Protocol (IP)
VPN: Virtual Private Network
WAAS: Wide Area Application Services
WAN: Wide area network
Wi-Fi: A trade name for the popular wireless technology used in home networks, mobile phones, video games and other electronic devices that require some form of wireless networking capability. Wi-Fi technologies are supported by nearly every modern personal computer operating system, most advanced game consoles and laptops, and many printers and other peripherals.
WOA: Web Platform Oriented Architecture

APPENDIX C ***- Code Examples***

MICROSOFT
Microsoft Developer Studio Visual C++ Client and Server (Protocol Independent)

Client (appropriately commented code provided below)
The following code is the IP agnostic, Protocol Independent (PI) Client.c file, which is an IPv6-enabled version of the Simplec.c file provided within the Microsoft SDK.

```
// client.c - Simple TCP/UDP client using Winsock 2.2
//
//      This is a part of the Microsoft<entity type="reg"/> Source Code Samples.
//      Copyright 1996 - 2000 Microsoft Corporation.
//      All rights reserved.
//      This source code is only intended as a supplement to
//      Microsoft Development Tools and/or WinHelp<entity type="reg"/> documentation.
//      See these sources for detailed information regarding the
//      Microsoft samples programs.
//
#define WIN32_LEAN_AND_MEAN

#include <winsock2.h>
#include <ws2tcpip.h>
#include <stdio.h>
#include <stdlib.h>
//#include <string.h>


// Needed for the Windows 2000 IPv6 Tech Preview.
#if (_WIN32_WINNT == 0x0500)
#include <tpipv6.h>
#endif

// Link with ws2_32.lib
#pragma comment(lib, "Ws2_32.lib")

#define STRICMP _stricmp


//
// This code assumes that at the transport level, the system only supports
```

```
// one stream protocol (TCP) and one datagram protocol (UDP).  Therefo
re,
// specifying a socket type of SOCK_STREAM is equivalent to specifying
TCP
// and specifying a socket type of SOCK_DGRAM is equivalent to specify
ing UDP.
//

#define DEFAULT_SERVER     NULL // Will use the loopback interface
#define DEFAULT_FAMILY     PF_UNSPEC    // Accept either IPv4 or IPv6
#define DEFAULT_SOCKTYPE   SOCK_STREAM  // TCP
#define DEFAULT_PORT       "5001"       // Arbitrary, albiet a histori
cal test port
#define DEFAULT_EXTRA      0    // Number of "extra" bytes to send

#define BUFFER_SIZE        65536

#define UNKNOWN_NAME "<unknown>"

void Usage(char *ProgName)
{
    fprintf(stderr, "\nSimple socket sample client program.\n");
    fprintf(stderr,
            "\n%s [-s server] [-f family] [-t transport] [-p port] [-b
bytes] [-n number]\n\n",
            ProgName);
    fprintf(stderr, "  server\tServer name or IP address.  (default: %
s)\n",
            (DEFAULT_SERVER == NULL) ? "loopback address" : DEFAULT_SE
RVER);
    fprintf(stderr,
            "  family\tOne of PF_INET, PF_INET6 or PF_UNSPEC.  (defaul
t: %s)\n",
            (DEFAULT_FAMILY ==
             PF_UNSPEC) ? "PF_UNSPEC" : ((DEFAULT_FAMILY ==
                                          PF_INET) ? "PF_INET" : "PF_I
NET6"));
    fprintf(stderr, "  transport\tEither TCP or UDP.  (default: %s)\n"
,
            (DEFAULT_SOCKTYPE == SOCK_STREAM) ? "TCP" : "UDP");
    fprintf(stderr, "  port\t\tPort on which to connect.  (default: %s
)\n",
            DEFAULT_PORT);
    fprintf(stderr, "  bytes\t\tBytes of extra data to send.  (default
: %d)\n",
```

```
            DEFAULT_EXTRA);
    fprintf(stderr, "  number\tNumber of sends to perform.  (default:
1)\n");
    fprintf(stderr, "  (-n by itself makes client run in an infinite l
oop,");
    fprintf(stderr, " Hit Ctrl-C to terminate)\n");
    WSACleanup();
    exit(1);
}

LPTSTR PrintError(int ErrorCode)
{
    static TCHAR Message[1024];

    // If this program was multithreaded, we'd want to use
    // FORMAT_MESSAGE_ALLOCATE_BUFFER instead of a static buffer here.
    // (And of course, free the buffer when we were done with it)

    FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_IGNORE_I
NSERTS |
                  FORMAT_MESSAGE_MAX_WIDTH_MASK,
                  NULL, ErrorCode, MAKELANGID(LANG_NEUTRAL, SUBLANG_DE
FAULT),
                  Message, 1024, NULL);
    return Message;
}

int ReceiveAndPrint(SOCKET ConnSocket, char *Buffer, int BufLen)
{
    int AmountRead;

    AmountRead = recv(ConnSocket, Buffer, BufLen, 0);
    if (AmountRead == SOCKET_ERROR) {
        fprintf(stderr, "recv() failed with error %d: %s\n",
                WSAGetLastError(), PrintError(WSAGetLastError()));
        closesocket(ConnSocket);
        WSACleanup();
        exit(1);
    }
    //
    // We are not likely to see this with UDP, since there is no
    // 'connection' established.
    //
    if (AmountRead == 0) {
        printf("Server closed connection\n");
```

```
        closesocket(ConnSocket);
        WSACleanup();
        exit(0);
    }

    printf("Received %d bytes from server: [%.*s]\n",
           AmountRead, AmountRead, Buffer);

    return AmountRead;
}

int main(int argc, char **argv)
{

    char Buffer[BUFFER_SIZE], AddrName[NI_MAXHOST];

    char *Server = DEFAULT_SERVER;
    int Family = DEFAULT_FAMILY;
    int SocketType = DEFAULT_SOCKTYPE;
    char *Port = DEFAULT_PORT;

    WSADATA wsaData;

    int i, RetVal, AddrLen, AmountToSend;
    int ExtraBytes = DEFAULT_EXTRA;
    unsigned int Iteration, MaxIterations = 1;
    BOOL RunForever = FALSE;

    ADDRINFO Hints, *AddrInfo, *AI;
    SOCKET ConnSocket = INVALID_SOCKET;
    struct sockaddr_storage Addr;

    if (argc > 1) {
        for (i = 1; i < argc; i++) {
            if (((argv[i][0] == '-') || (argv[i][0] == '/')) &&
                (argv[i][1] != 0) && (argv[i][2] == 0)) {
                switch (tolower(argv[i][1])) {
                case 'f':
                    if (!argv[i + 1])
                        Usage(argv[0]);
                    if (!STRICMP(argv[i + 1], "PF_INET"))
                        Family = PF_INET;
                    else if (!STRICMP(argv[i + 1], "AF_INET"))
                        Family = PF_INET;
                    else if (!STRICMP(argv[i + 1], "PF_INET6"))
```

```
            Family = PF_INET6;
        else if (!STRICMP(argv[i + 1], "AF_INET6"))
            Family = PF_INET6;
        else if (!STRICMP(argv[i + 1], "PF_UNSPEC"))
            Family = PF_UNSPEC;
        else if (!STRICMP(argv[i + 1], "AF_UNSPEC"))
            Family = PF_UNSPEC;
        else
            Usage(argv[0]);
        i++;
        break;

    case 't':
        if (!argv[i + 1])
            Usage(argv[0]);
        if (!STRICMP(argv[i + 1], "TCP"))
            SocketType = SOCK_STREAM;
        else if (!STRICMP(argv[i + 1], "UDP"))
            SocketType = SOCK_DGRAM;
        else
            Usage(argv[0]);
        i++;
        break;

    case 's':
        if (argv[i + 1]) {
            if (argv[i + 1][0] != '-') {
                Server = argv[++i];
                break;
            }
        }
        Usage(argv[0]);
        break;

    case 'p':
        if (argv[i + 1]) {
            if (argv[i + 1][0] != '-') {
                Port = argv[++i];
                break;
            }
        }
        Usage(argv[0]);
        break;

    case 'b':
```

```
                    if (argv[i + 1]) {
                        if (argv[i + 1][0] != '-') {
                            ExtraBytes = atoi(argv[++i]);
                            if (ExtraBytes >
                                sizeof (Buffer) -
                                sizeof ("Message #4294967295"))
                                Usage(argv[0]);
                            break;
                        }
                    }
                    Usage(argv[0]);
                    break;

                case 'n':
                    if (argv[i + 1]) {
                        if (argv[i + 1][0] != '-') {
                            MaxIterations = atoi(argv[++i]);
                            break;
                        }
                    }
                    RunForever = TRUE;
                    break;

                default:
                    Usage(argv[0]);
                    break;
                }
            } else
                Usage(argv[0]);
        }
    }
    // Ask for Winsock version 2.2.
    if ((RetVal = WSAStartup(MAKEWORD(2, 2), &wsaData)) != 0) {
        fprintf(stderr, "WSAStartup failed with error %d: %s\n",
                RetVal, PrintError(RetVal));
        WSACleanup();
        return -1;
    }
    //
    // By not setting the AI_PASSIVE flag in the hints to getaddrinfo,
we're
    // indicating that we intend to use the resulting address(es) to c
onnect
    // to a service.  This means that when the Server parameter is NUL
L,
```

```
    // getaddrinfo will return one entry per allowed protocol family
    // containing the loopback address for that family.
    //

    memset(&Hints, 0, sizeof (Hints));
    Hints.ai_family = Family;
    Hints.ai_socktype = SocketType;
    RetVal = getaddrinfo(Server, Port, &Hints, &AddrInfo);
    if (RetVal != 0) {
        fprintf(stderr,
                "Cannot resolve address [%s] and port [%s], error %d:
%s\n",
                Server, Port, RetVal, gai_strerror(RetVal));
        WSACleanup();
        return -1;
    }
    //
    // Try each address getaddrinfo returned, until we find one to whi
ch
    // we can successfully connect.
    //
    for (AI = AddrInfo; AI != NULL; AI = AI->ai_next) {

        // Open a socket with the correct address family for this addr
ess.

        ConnSocket = socket(AI->ai_family, AI->ai_socktype, AI->ai_pro
tocol);

//**** DEBUG
        printf("socket call with family: %d socktype: %d, protocol: %d
\n",
               AI->ai_family, AI->ai_socktype, AI->ai_protocol);
        if (ConnSocket == INVALID_SOCKET)
            printf("socket call failed with %d\n", WSAGetLastError());
//**** DEBUG END

        if (ConnSocket == INVALID_SOCKET) {
            fprintf(stderr, "Error Opening socket, error %d: %s\n",
                    WSAGetLastError(), PrintError(WSAGetLastError()));
            continue;
        }
        //
        // Notice that nothing in this code is specific to whether we
        // are using UDP or TCP.
```

```
        //
        // When connect() is called on a datagram socket, it does not
        // actually establish the connection as a stream (TCP) socket
        // would. Instead, TCP/IP establishes the remote half of the
        // (LocalIPAddress, LocalPort, RemoteIP, RemotePort) mapping.
        // This enables us to use send() and recv() on datagram socket
s,
        // instead of recvfrom() and sendto().
        //

        printf("Attempting to connect to: %s\n", Server ? Server : "lo
calhost");
        if (connect(ConnSocket, AI->ai_addr, (int) AI->ai_addrlen) !=
SOCKET_ERROR)
            break;

        i = WSAGetLastError();
        if (getnameinfo(AI->ai_addr, (int) AI->ai_addrlen, AddrName,
                        sizeof (AddrName), NULL, 0, NI_NUMERICHOST) !=
0)
            strcpy_s(AddrName, sizeof (AddrName), UNKNOWN_NAME);
        fprintf(stderr, "connect() to %s failed with error %d: %s\n",
                AddrName, i, PrintError(i));
        closesocket(ConnSocket);
    }

    if (AI == NULL) {
        fprintf(stderr, "Fatal error: unable to connect to the server.
\n");
        WSACleanup();
        return -1;
    }
    //
    // This demonstrates how to determine to where a socket is connect
ed.
    //
    AddrLen = sizeof (Addr);
    if (getpeername(ConnSocket, (LPSOCKADDR) & Addr, (int *) &AddrLen)
== SOCKET_ERROR) {
        fprintf(stderr, "getpeername() failed with error %d: %s\n",
                WSAGetLastError(), PrintError(WSAGetLastError()));
    } else {
        if (getnameinfo((LPSOCKADDR) & Addr, AddrLen, AddrName,
                        sizeof (AddrName), NULL, 0, NI_NUMERICHOST) !=
0)
```

```
            strcpy_s(AddrName, sizeof (AddrName), UNKNOWN_NAME);
        printf("Connected to %s, port %d, protocol %s, protocol family
%s\n",
                AddrName, ntohs(SS_PORT(&Addr)),
                (AI->ai_socktype == SOCK_STREAM) ? "TCP" : "UDP",
                (AI->ai_family == PF_INET) ? "PF_INET" : "PF_INET6");
    }

    // We are done with the address info chain, so we can free it.
    freeaddrinfo(AddrInfo);

    //
    // Find out what local address and port the system picked for us.
    //
    AddrLen = sizeof (Addr);
    if (getsockname(ConnSocket, (LPSOCKADDR) & Addr, &AddrLen) == SOCK
ET_ERROR) {
        fprintf(stderr, "getsockname() failed with error %d: %s\n",
                WSAGetLastError(), PrintError(WSAGetLastError()));
    } else {
        if (getnameinfo((LPSOCKADDR) & Addr, AddrLen, AddrName,
                        sizeof (AddrName), NULL, 0, NI_NUMERICHOST) !=
0)
            strcpy_s(AddrName, sizeof (AddrName), UNKNOWN_NAME);
        printf("Using local address %s, port %d\n",
               AddrName, ntohs(SS_PORT(&Addr)));
    }

    //
    // Send and receive in a loop for the requested number of iteratio
ns.
    //
    for (Iteration = 0; RunForever || Iteration < MaxIterations; Itera
tion++) {

        // Compose a message to send.
        AmountToSend =
            sprintf_s(Buffer, sizeof (Buffer), "Message #%u", Iteratio
n + 1);
        for (i = 0; i < ExtraBytes; i++) {
            Buffer[AmountToSend++] = (char) ((i & 0x3f) + 0x20);
        }

        // Send the message.  Since we are using a blocking socket, th
is
```

```
        // call shouldn't return until it's able to send the entire am
ount.
        RetVal = send(ConnSocket, Buffer, AmountToSend, 0);
        if (RetVal == SOCKET_ERROR) {
            fprintf(stderr, "send() failed with error %d: %s\n",
                    WSAGetLastError(), PrintError(WSAGetLastError()));
            WSACleanup();
            return -1;
        }

        printf("Sent %d bytes (out of %d bytes) of data: [%.*s]\n",
               RetVal, AmountToSend, AmountToSend, Buffer);

        // Clear buffer just to prove we're really receiving something
.
        memset(Buffer, 0, sizeof (Buffer));

        // Receive and print server's reply.
        ReceiveAndPrint(ConnSocket, Buffer, sizeof (Buffer));
    }

    // Tell system we're done sending.
    printf("Done sending\n");
    shutdown(ConnSocket, SD_SEND);

    //
    // Since TCP does not preserve message boundaries, there may still
    // be more data arriving from the server.  So we continue to recei
ve
    // data until the server closes the connection.
    //
    if (SocketType == SOCK_STREAM)
        while (ReceiveAndPrint(ConnSocket, Buffer, sizeof (Buffer)) !=
0) ;

    closesocket(ConnSocket);
    WSACleanup();
    return 0;
}
```

Server (appropriately commented code provided below)

The following code is the IP-agnostic PI Server.c file, which is an IPv6-enabled version of the Simples.c file

```
//
// server.c - Simple TCP/UDP server using Winsock 2.2
//
```

```
//      This is a part of the Microsoft<entity type="reg"/> Source Cod
e Samples.
//      Copyright 1996 - 2000 Microsoft Corporation.
//      All rights reserved.
//      This source code is only intended as a supplement to
//      Microsoft Development Tools and/or WinHelp<entity type="reg"/>
documentation.
//      See these sources for detailed information regarding the
//      Microsoft samples programs.
//

#undef UNICODE

#define WIN32_LEAN_AND_MEAN

#include <windows.h>
#include <winsock2.h>
#include <ws2tcpip.h>
#include <mstcpip.h>
#include <stdlib.h>
#include <stdio.h>

// Needed for the Windows 2000 IPv6 Tech Preview.
#if (_WIN32_WINNT == 0x0500)
#include <tpipv6.h>
#endif

// Link with ws2_32.lib
#pragma comment(lib, "Ws2_32.lib")

#define STRICMP _stricmp

//
// This code assumes that at the transport level, the system only supp
orts
// one stream protocol (TCP) and one datagram protocol (UDP).  Therefo
re,
// specifying a socket type of SOCK_STREAM is equivalent to specifying
TCP
// and specifying a socket type of SOCK_DGRAM is equivalent to specify
ing UDP.
//

#define DEFAULT_FAMILY     PF_UNSPEC    // Accept either IPv4 or IPv6
#define DEFAULT_SOCKTYPE   SOCK_STREAM  // TCP
```

```
#define DEFAULT_PORT       "5001"       // Arbitrary, albiet a histori
cal test port
#define BUFFER_SIZE        64   // Set very small for demonstration pu
rposes

void Usage(char *ProgName)
{
    fprintf(stderr, "\nSimple socket sample server program.\n");
    fprintf(stderr,
            "\n%s [-f family] [-t transport] [-p port] [-a address]\n\
n",
            ProgName);
    fprintf(stderr,
            "  family\tOne of PF_INET, PF_INET6 or PF_UNSPEC.  (defaul
t %s)\n",
            (DEFAULT_FAMILY ==
             PF_UNSPEC) ? "PF_UNSPEC" : ((DEFAULT_FAMILY ==
                                          PF_INET) ? "PF_INET" : "PF_I
NET6"));
    fprintf(stderr, "  transport\tEither TCP or UDP.  (default: %s)\n"
,
            (DEFAULT_SOCKTYPE == SOCK_STREAM) ? "TCP" : "UDP");
    fprintf(stderr, "  port\t\tPort on which to bind.  (default %s)\n"
,
            DEFAULT_PORT);
    fprintf(stderr,
            "  address\tIP address on which to bind.  (default: unspec
ified address)\n");
    WSACleanup();
    exit(1);
}

LPSTR PrintError(int ErrorCode)
{
    static char Message[1024];

    // If this program was multithreaded, we'd want to use
    // FORMAT_MESSAGE_ALLOCATE_BUFFER instead of a static buffer here.
    // (And of course, free the buffer when we were done with it)

    FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_IGNORE_I
NSERTS |
                  FORMAT_MESSAGE_MAX_WIDTH_MASK, NULL, ErrorCode,
                  MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
                  (LPSTR) Message, 1024, NULL);
```

```
    return Message;
}

int main(int argc, char **argv)
{
    char Buffer[BUFFER_SIZE], Hostname[NI_MAXHOST];
    int Family = DEFAULT_FAMILY;
    int SocketType = DEFAULT_SOCKTYPE;
    char *Port = DEFAULT_PORT;
    char *Address = NULL;
    int i, NumSocks, RetVal, FromLen, AmountRead;
//    int idx;
    SOCKADDR_STORAGE From;
    WSADATA wsaData;
    ADDRINFO Hints, *AddrInfo, *AI;
    SOCKET ServSock[FD_SETSIZE];
    fd_set SockSet;

    // Parse arguments
    if (argc > 1) {
        for (i = 1; i < argc; i++) {
            if ((argv[i][0] == '-') || (argv[i][0] == '/') &&
                (argv[i][1] != 0) && (argv[i][2] == 0)) {
                switch (tolower(argv[i][1])) {
                case 'f':
                    if (!argv[i + 1])
                        Usage(argv[0]);
                    if (!STRICMP(argv[i + 1], "PF_INET"))
                        Family = PF_INET;
                    else if (!STRICMP(argv[i + 1], "PF_INET6"))
                        Family = PF_INET6;
                    else if (!STRICMP(argv[i + 1], "PF_UNSPEC"))
                        Family = PF_UNSPEC;
                    else
                        Usage(argv[0]);
                    i++;
                    break;

                case 't':
                    if (!argv[i + 1])
                        Usage(argv[0]);
                    if (!STRICMP(argv[i + 1], "TCP"))
                        SocketType = SOCK_STREAM;
                    else if (!STRICMP(argv[i + 1], "UDP"))
                        SocketType = SOCK_DGRAM;
```

```
                else
                    Usage(argv[0]);
                i++;
                break;

            case 'a':
                if (argv[i + 1]) {
                    if (argv[i + 1][0] != '-') {
                        Address = argv[++i];
                        break;
                    }
                }
                Usage(argv[0]);
                break;

            case 'p':
                if (argv[i + 1]) {
                    if (argv[i + 1][0] != '-') {
                        Port = argv[++i];
                        break;
                    }
                }
                Usage(argv[0]);
                break;

            default:
                Usage(argv[0]);
                break;
            }
        } else
            Usage(argv[0]);
    }
}
// Ask for Winsock version 2.2.
if ((RetVal = WSAStartup(MAKEWORD(2, 2), &wsaData)) != 0) {
    fprintf(stderr, "WSAStartup failed with error %d: %s\n",
            RetVal, PrintError(RetVal));
    WSACleanup();
    return -1;
}

if (Port == NULL) {
    Usage(argv[0]);
}
//
```

```
    // By setting the AI_PASSIVE flag in the hints to getaddrinfo, we'
re
    // indicating that we intend to use the resulting address(es) to b
ind
    // to a socket(s) for accepting incoming connections.  This means
that
    // when the Address parameter is NULL, getaddrinfo will return one
    // entry per allowed protocol family containing the unspecified ad
dress
    // for that family.
    //
    memset(&Hints, 0, sizeof (Hints));
    Hints.ai_family = Family;
    Hints.ai_socktype = SocketType;
    Hints.ai_flags = AI_NUMERICHOST | AI_PASSIVE;
    RetVal = getaddrinfo(Address, Port, &Hints, &AddrInfo);
    if (RetVal != 0) {
        fprintf(stderr, "getaddrinfo failed with error %d: %s\n",
                RetVal, gai_strerror(RetVal));
        WSACleanup();
        return -1;
    }
    //
    // For each address getaddrinfo returned, we create a new socket,
    // bind that address to it, and create a queue to listen on.
    //
    for (i = 0, AI = AddrInfo; AI != NULL; AI = AI->ai_next) {

        // Highly unlikely, but check anyway.
        if (i == FD_SETSIZE) {
            printf("getaddrinfo returned more addresses than we could
use.\n");
            break;
        }
        // This example only supports PF_INET and PF_INET6.
        if ((AI->ai_family != PF_INET) && (AI->ai_family != PF_INET6))
            continue;

        // Open a socket with the correct address family for this addr
ess.
        ServSock[i] = socket(AI->ai_family, AI->ai_socktype, AI->ai_pr
otocol);
        if (ServSock[i] == INVALID_SOCKET) {
            fprintf(stderr, "socket() failed with error %d: %s\n",
                    WSAGetLastError(), PrintError(WSAGetLastError()));
```

```
            continue;
        }

        if ((AI->ai_family == PF_INET6) &&
            IN6_IS_ADDR_LINKLOCAL((IN6_ADDR *) INETADDR_ADDRESS(AI->ai
_addr)) &&
            (((SOCKADDR_IN6 *) (AI->ai_addr))->sin6_scope_id == 0)
            ) {
            fprintf(stderr,
                    "IPv6 link local addresses should specify a scope
ID!\n");
        }
        //
        // bind() associates a local address and port combination
        // with the socket just created. This is most useful when
        // the application is a server that has a well-known port
        // that clients know about in advance.
        //
        if (bind(ServSock[i], AI->ai_addr, (int) AI->ai_addrlen) == SO
CKET_ERROR) {
            fprintf(stderr, "bind() failed with error %d: %s\n",
                    WSAGetLastError(), PrintError(WSAGetLastError()));
            closesocket(ServSock[i]);
            continue;
        }
        //
        // So far, everything we did was applicable to TCP as well as
UDP.
        // However, there are certain fundamental differences between
stream
        // protocols such as TCP and datagram protocols such as UDP.
        //
        // Only connection orientated sockets, for example those of ty
pe
        // SOCK_STREAM, can listen() for incoming connections.
        //
        if (SocketType == SOCK_STREAM) {
            if (listen(ServSock[i], 5) == SOCKET_ERROR) {
                fprintf(stderr, "listen() failed with error %d: %s\n",
                        WSAGetLastError(), PrintError(WSAGetLastError(
)));
                closesocket(ServSock[i]);
                continue;
            }
        }
```

```
        printf("'Listening' on port %s, protocol %s, protocol family %
s\n",
               Port, (SocketType == SOCK_STREAM) ? "TCP" : "UDP",
               (AI->ai_family == PF_INET) ? "PF_INET" : "PF_INET6");
        i++;
    }

    freeaddrinfo(AddrInfo);

    if (i == 0) {
        fprintf(stderr, "Fatal error: unable to serve on any address.\
n");
        WSACleanup();
        return -1;
    }
    NumSocks = i;

    //
    // We now put the server into an eternal loop,
    // serving requests as they arrive.
    //
    FD_ZERO(&SockSet);
    while (1) {

        FromLen = sizeof (From);

        //
        // For connection orientated protocols, we will handle the
        // packets received from a connection collectively.  For datag
ram
        // protocols, we have to handle each datagram individually.
        //

        //
        // Check to see if we have any sockets remaining to be served
        // from previous time through this loop.  If not, call select(
)
        // to wait for a connection request or a datagram to arrive.
        //
        for (i = 0; i < NumSocks; i++) {
            if (FD_ISSET(ServSock[i], &SockSet))
                break;
        }
        if (i == NumSocks) {
```

```
            for (i = 0; i < NumSocks; i++)
                FD_SET(ServSock[i], &SockSet);
            if (select(NumSocks, &SockSet, 0, 0, 0) == SOCKET_ERROR) {
                fprintf(stderr, "select() failed with error %d: %s\n",
                        WSAGetLastError(), PrintError(WSAGetLastError(
)));
                WSACleanup();
                return -1;
            }
        }
        for (i = 0; i < NumSocks; i++) {
            if (FD_ISSET(ServSock[i], &SockSet)) {
                FD_CLR(ServSock[i], &SockSet);
                break;
            }
        }

        if (SocketType == SOCK_STREAM) {
            SOCKET ConnSock;

            //
            // Since this socket was returned by the select(), we know
we
            // have a connection waiting and that this accept() won't
block.
            //
            ConnSock = accept(ServSock[i], (LPSOCKADDR) & From, &FromL
en);
            if (ConnSock == INVALID_SOCKET) {
                fprintf(stderr, "accept() failed with error %d: %s\n",
                        WSAGetLastError(), PrintError(WSAGetLastError(
)));
                WSACleanup();
                return -1;
            }
            if (getnameinfo((LPSOCKADDR) & From, FromLen, Hostname,
                            sizeof (Hostname), NULL, 0, NI_NUMERICHOST
) != 0)
                strcpy_s(Hostname, NI_MAXHOST, "<unknown>");
            printf("\nAccepted connection from %s\n", Hostname);

            //
            // This sample server only handles connections sequentiall
y.
            // To handle multiple connections simultaneously, a server
```

```
            // would likely want to launch another thread or process a
t this
            // point to handle each individual connection.  Alternativ
ely,
            // it could keep a socket per connection and use select()
            // on the fd_set to determine which to read from next.
            //
            // Here we just loop until this connection terminates.
            //

            while (1) {

                //
                // We now read in data from the client.  Because TCP
                // does NOT maintain message boundaries, we may recv()
                // the client's data grouped differently than it was
                // sent.  Since all this server does is echo the data
it
                // receives back to the client, we don't need to conce
rn
                // ourselves about message boundaries.  But it does me
an
                // that the message data we print for a particular rec
v()
                // below may contain more or less data than was contai
ned
                // in a particular client send().
                //

                AmountRead = recv(ConnSock, Buffer, sizeof (Buffer), 0
);
                if (AmountRead == SOCKET_ERROR) {
                    fprintf(stderr, "recv() failed with error %d: %s\n
",
                            WSAGetLastError(), PrintError(WSAGetLastEr
ror()));
                    closesocket(ConnSock);
                    break;
                }
                if (AmountRead == 0) {
                    printf("Client closed connection\n");
                    closesocket(ConnSock);
                    break;
                }
```

```
                printf("Received %d bytes from client: [%.*s]\n",
                       AmountRead, AmountRead, Buffer);
                printf("Echoing same data back to client\n");

                RetVal = send(ConnSock, Buffer, AmountRead, 0);
                if (RetVal == SOCKET_ERROR) {
                    fprintf(stderr, "send() failed: error %d: %s\n",
                            WSAGetLastError(), PrintError(WSAGetLastEr
ror()));
                    closesocket(ConnSock);
                    break;
                }
            }

        } else {

            //
            // Since UDP maintains message boundaries, the amount of d
ata
            // we get from a recvfrom() should match exactly the amoun
t of
            // data the client sent in the corresponding sendto().
            //
            AmountRead = recvfrom(ServSock[i], Buffer, sizeof (Buffer)
, 0,
                                  (LPSOCKADDR) & From, &FromLen);
            if (AmountRead == SOCKET_ERROR) {
                fprintf(stderr, "recvfrom() failed with error %d: %s\n
",
                        WSAGetLastError(), PrintError(WSAGetLastError(
)));
                closesocket(ServSock[i]);
                break;
            }
            if (AmountRead == 0) {
                // This should never happen on an unconnected socket,
but...
                printf("recvfrom() returned zero, aborting\n");
                closesocket(ServSock[i]);
                break;
            }

            RetVal = getnameinfo((LPSOCKADDR) & From, FromLen, Hostnam
e,
```

```
                                    sizeof (Hostname), NULL, 0, NI_NUMERI
CHOST);
            if (RetVal != 0) {
                fprintf(stderr, "getnameinfo() failed with error %d: %
s\n",
                        RetVal, PrintError(RetVal));
                strcpy_s(Hostname, NI_MAXHOST, "<unknown>");
            }

            printf("Received a %d byte datagram from %s: [%.*s]\n",
                   AmountRead, Hostname, AmountRead, Buffer);
            printf("Echoing same data back to client\n");

            RetVal = sendto(ServSock[i], Buffer, AmountRead, 0,
                            (LPSOCKADDR) & From, FromLen);
            if (RetVal == SOCKET_ERROR) {
                fprintf(stderr, "send() failed with error %d: %s\n",
                        WSAGetLastError(), PrintError(WSAGetLastError(
)));
            }
        }
    }

    return 0;
}
```

Testing Service Testing Guide
October 2011
Version 1.6

Revision History

Date	Version	Description	Author
7/10/09	1.1	Added version number. Section 3.5 clarified to remove references to Field Test Release. Section 3.6 removed. The activities in that section were replaced by the EDM process outside of TS. Table of Contents updated.	Testing Service
7/10/10	1.2	Changed User Acceptance Test to User Functionality Test	OED Process Management Service
8/05/10	1.3	Removed OED from guide	Process Management
10/14/10	1.4	Repaired broken link to Testing QA Checklist	Process Management
08/23/11	1.5	Changed Operational Readiness Testing (ORT) to Operational Readiness Review (ORR)	Process Management
09/17/11	1.6	Deleted Testing QA Checklist as an obsolete artifact	Process Management

Table of Contents

1. PURPOSE XXX

2. RESPONSIBILITIES XXX

2.1. DEVELOPMENT MANAGER XXX
2.2. PROJECT TEAM XXXI
2.3. TESTING SERVICE XXXII

3. STEPS XXXIII

3.1. PERFORM PREPARATORY ACTIVITIES DURING PRODUCT BUILD XXXIII
3.2. CONDUCT TESTING SERVICE FOCUS MEETING XXXV
3.3. CONDUCT INDEPENDENTTESTING ACTIVITIES, INITIAL BUILD TESTING XXXVI
3.4. CONDUCT INDEPENDENT TESTING ACTIVITIES, SUBSEQUENT BUILD TESTING XXXVII
3.5. GRANT TESTING SERVICE APPROVAL TO PROCEED XXXIX

4. PROCESSES AND RELATED REFERENCES XXXIX

4.1. RELATED PROCESSES XXXIX
4.2. RELATED REFERENCES XL

1 Purpose

This Testing Service Testing Process Guide describes the purpose, roles, responsibilities, and steps for Testing Service (TS) approval and/or certification of application projects. The processes, roles, responsibilities, and steps detailed in this guide are specific and relevant to Testing Service testing. An application project is defined as consisting of applications as well as their associated services and related dependencies. While the application project may be reliant on these other services and software dependencies, the approval and/or certification given by TS is specific to the application in the context as the application relates to those relevant services and dependencies.

All steps, unless otherwise mentioned, are performed by the Development Manager (formerly known as Project Manager), Project Team, and Testing Service, which includes internal TS teams.

2 Responsibilities

The following roles and responsibilities are involved during the testing process conducted by TS.

2.1 *Development Manager*

The Development Manager is responsible for:

- Releasing the overall application project into the production environment through Product Support or Enterprise System Management (ESM)
- Managing the software development process for the application project
- Ensuring TS involvement in the project kick-off meeting, the two-day high level meeting with Office of Information (OI)-wide stakeholder representation, and any subsequent planning meeting(s) to discuss the implementation of the application project
- Coordinating with TS to have a Capacity Planning (CP) preliminary and final Application Self-Scoring Evaluation Support System (ASSESS) form submitted. The preliminary and final ASSESS form for the project can be obtained at http://vista.med.va.gov/capman/ASSESS_process.asp
- Completing and submitting the Testing Service Request System (TSRS) form necessary to initiate the initial request. For more information click the following link to access the system: http://vista.med.va.gov/testingservices/requests_tsrs.asp
- Soliciting from TS an initial estimate of the amount of time and effort for the testing tasks and adding these estimates to the overall project timeline schedule within the constraints imposed by the TS workload schedule calendar

NOTE: *The proposed dates for EIE Operational Readiness Review (ORR) and Initial Operating Capability (IOC) Testing this early in the planning stages will be further refined during a subsequent high level meeting with stakeholder representation.*

- Designating appropriate Project Team members as specified points of contact (POCs) to provide guidance and support throughout the independent testing process with TS
- Coordinating support from Project Team and dependency application(s)/service(s) and/or their components throughout the testing process

- Providing all appropriate software builds and full documentation of the application project as well as dependency application(s)/service(s) and/or their components, while adhering to the proper versioning procedure required by TS
- Ensuring that all software builds are viable and ready for testing, that the Product Build Quality Gate Review is satisfied before entering TS, and that the application has received Software Quality Assurance (SQA) certification. Viable builds are defined as being the application project with its associated dependency application(s)/service(s) and/or their components and all artifacts required in the Testing Service Entrance Criteria
- Accepting refined task estimates from TS as the system is further technically defined and incorporating these estimates to the master project schedule for the application project
- Ensuring all required TS Entrance Criteria are met as defined in Testing Service Entrance Criteria
- Keeping TS informed of all project schedule changes/variances and changes to application project build(s) that may necessitate scheduling changes to the Testing Service workload schedule calendar
- Receiving and coordinating status updates as obtained from TS throughout the testing and analysis process with Project Team members and all other stakeholders, such as dependency application(s) project team(s)

2.2 Project Team

The Project team is responsible for:

- Completing the preliminary and final ASSESS process as required for an application project comparison analysis. The preliminary and final ASSESS form for the project can be obtained at http://vista.med.va.gov/capman/ASSESS_process.asp
- Completing the appropriate ProPath processes, including Product Component Test, Component Integration Test, System Test, and User Functionality Test
- Delivering all required input artifacts as referenced in Testing Service Entrance Criteria
- Ensuring the requirements to be tested within the selected features, functions, and/or use cases are testable, unambiguous, and ready for testing before entering Independent Testing
- Providing guidance and support to TS throughout the Approval and/or Certification process
- Providing all appropriate software builds of the application project, as well as dependency application(s)/service(s) and/or their components, as one complete package while adhering to the proper versioning procedure required by TS
- Ensuring that all builds, in the complete package delivered to TS, are viable and ready for testing before entering TS and have received SQA certification. Builds are defined as being the application project with its associated dependency application(s)/service(s) and/or their components

NOTE: *For VistA-Legacy (Waterfall) development, complete the* SQA Review Checklist.

- Providing assistance with the installation in the Test Center of the System Under Test (SUT) including all associated databases with appropriate data volume required for testing
- Providing a demonstration of the SUT to TS as the application project is installed in the Test Center
- Providing timely response to system architectural and other technical questions as needed by TS
- Receiving and acting upon identified defects reported by TS

- Coordinating all escalation processes in conjunction with TS
- Determining, documenting, and publishing the criteria, in collaboration with TS, which the SUT must meet for TS formal approval to proceed and for formal certification for national deployment

2.3 Testing Service

Testing Service is responsible for:

- Reviewing, with the Development Manager and the Project Team, all TS high level processes for IV&V and Performance Engineering testing throughout all necessary points of the Testing Service HeV Certification Lifecycle
- Reviewing, with the Development Manager and the Project Team, the TS Entrance Criteria artifacts as submitted by the Development Manager and the Project Team and defined in the Testing Service Entrance Criteria
- Providing an evaluation score based on the data submitted by the Project Team in preliminary and final ASSESS forms

NOTE: *Refer to http://vista.med.va.gov/capman/ASSESS_process.asp for details on the Capacity Planning ASSESS process.*

- Gathering current system performance statistics by providing workload baselines for the Project Team's application project using applicable metrics when available
- Coordinating an environment review of the hardware and software components including any dependency application(s)/service(s) and/or their components that comprise the SUT
- Ensuring all appropriate builds of the application project as well as dependency application(s)/service(s) and/or their components as provided by the Project Team adhere to the proper versioning procedure required by TS
- Verifying that all builds delivered from the Project Team as one complete package are viable and ready for TS testing before entering TS and have received SQA certification. Builds are defined as being the application project with its associated dependency application(s)/service(s) and/or their components

NOTE: *For VistA-Legacy (Waterfall) development, complete the* SQA Review Checklist.

- Assisting in the setup of the hardware servers, network components, operating system software, application servers (such as Web Logic), and database(s) (such as Oracle, Cache) for the test environment(s)
- Verifying with the Project Team any features, functions and/or use cases to be validation and performance tested by TS
- Coordinating all Testing Service testing activities as conducted by TS
- Developing, reviewing, approving, and distributing Test Plans and managing the testing project according to the stated plans
- Performing all Testing Service testing activities as defined in all Test Plans as agreed upon between the Development Manager, the Project Team, and TS
- Writing and executing manual and automated test scripts designed to test the features, functions, and use cases selected for the SUT
- Ensuring all TS testing output artifacts designated as configuration items are complete, accurate, and stored in a version controlled manner within TS

- Developing, reviewing, approving, and distributing the various result reports as referenced in Testing Service Reporting Analyses such as the Architecture Validation Analysis (AVA), Performance Test Analysis (PTA), and/or IV&V Test Analysis Reports as needed
- Providing TS Approval to proceed at the completion of Testing Service Testing

3 Steps

3.1 Perform Preparatory Activities During Product Build

3.1.1 Project Team

- Submits a Service Request using the Testing Service Request System (TSRS) available by following this link: http://vista.med.va.gov/testingservices/requests_tsrs.asp

NOTE: *When submitting the TSRS, the Development Manager attaches the required artifacts.*

- Completes and submits the Preliminary ASSESS form from the CP web page located at http://vista.med.va.gov/capman/ASSESS_process.asp.

3.1.2 Testing Service

- Reviews the TS Request (TSR) Form submitted by the Project Team.
- Reviews all CP processes as defined in Testing Service Capacity Planning Processes to prepare the deliverables for the project that are necessary to scope the impact from a performance and capacity perspective.
- Reviews and scores the Preliminary ASSESS form (Capacity Planning).

3.1.3 Project Team

- Reviews the TS Entrance Criteria to ensure compliance.
- Conducts subsequent focused project calls with TS and other appropriate stakeholders [e.g., Application Structure and Integration Service (ASIS) (http://vista.med.va.gov/asis/default.htm), Technology Evaluation Team (TET), Technology Modernization Project (TMP)] to discuss specifics relating to the Elaboration Phase and future engagement flow activities.
- Provides any updated information identified since the initial project kick-off meeting such as:
  1. Modified application project schedule with focus on proposed dates for EIE Operational Readiness Review (ORR) and Initial Operating Capability (IOC) testing, to be further refined during a subsequent high level meeting with OI-wide stakeholder representation.
  2. Modified IRA, Stakeholder Request, or Vision Document and any other pertinent documentation.
  3. Modifications to the project's intended architecture, business workload, scope, dependencies, and project milestones.
  4. Changes to the most critical use-cases and workflow, and proposed plans for rehosting/re-engineering those use-cases.
  5. Modification of key representatives to be involved throughout the testing process lifecycle.
- Updates ASSESS information in a draft of the Final ASSESS form. The form is found on the CP Web page located at http://vista.med.va.gov/capman/ASSESS_process.asp

3.1.4 Testing Service

- Ensures that necessary key representatives (from TS perspective) attend subsequent focused application project calls and subsequent meetings regarding the project.
- Reviews all Project Team TS Entrance Criteria as referenced in the Testing Service Entrance Criteria that must be satisfied before acceptance into Testing Service for formal product review.
- Reviews TS Entrance Criteria as referenced in the Testing Service Entrance Criteria to provide information necessary for entry into TS.
- Reviews all CP Processes as defined in Testing Service Capacity Planning Processes to prepare the deliverables for the project that are necessary to scope the impact from a performance and capacity perspective.
- Acknowledges submitted Service Request and expands to include all relevant project artifacts, including proposed entry dates into Testing Service for approval and/or certification testing process.

3.1.5 Project Team

- Completes and submits the Final ASSESS form from the CP web page located at http://vista.med.va.gov/capman/ASSESS_process.asp
- Performs Product Component Test, Component Integration Test, System Test, and User Functionality Test and provides the test results.
- Ensures that all final application builds are viable, have received SQA certification, and are ready for testing before entering TS. Viable builds are defined as being the application project with its associated dependency application(s)/service(s) and/or their components and all artifacts required in the Testing Service Entrance Criteria.

NOTE: *For VistA-Legacy (Waterfall) development, complete the* SQA Review Checklist.

3.1.6 Testing Service

- Ensures that key representatives attend all subsequent focused application project kick-off calls.
- Determines roles and responsibilities of Testing Service and SUT personnel for all activities.
- Reviews and scores the Final ASSESS form (Capacity Planning).
- Reviews all documentation for the SUT and dependency application(s)/service(s) and/or their components.
- Analyzes the requirements necessary to install the databases for the SUT in the Test Center. This includes determination of the system to be used, size, method or technique, and responsibility for populating all VistA, Cache, Oracle, SQL Server, and/or other databases used by the SUT.
    1. Determine volumes of data necessary to support the performance testing for all databases used by the SUT. Use the Package Resource Evaluation (PRE), Package Event Modeling Analysis (PEMA) and Package User Modeling Analysis (PUMA) reports as needed. PRE, PEMA and PUMA reports are defined in Testing Service Capacity Planning Processes.
    2. Determine methods for populating databases with additional data volume to support the performance tests.
- Analyzes the system environment requirements necessary for installation of the SUT.
- Defines approach for both IV&V and Performance Engineering testing activities.

- Prepares IV&V Test Plan and/or Performance Test Plan (PTP) in collaboration with Project Team and all stakeholders on objectives, schedule, and expectations of the engagement. Project plans are binding between TS and Project Team.
- Sets up Test Director Project for the SUT using IV&V Test Director Standards and Conventions as a guideline for processes and naming conventions. Test Director is a testing management tool designed to manage testing activities from requirements analysis through SUT, planning of tests, packaging and execution of automated and manual tests, and tracking defects to completion.
- Builds the environment utilizing installation guides provided by the Project Team for the application project as well as dependency application(s)/service(s) and/or their components.

3.2 Conduct Testing Service Focus Meeting

3.2.1 Development Manager/Project Team

- Holds Focus Meeting with TS.
- Delivers all required input artifacts as referenced in the Testing Service Entrance Criteria.
- Provides guidance and support to TS throughout the Approval and/or Certification process.
- Ensures that the SUT has successfully completed SQA
- Provides all appropriate final application builds, as one complete package, of the application project as well as dependency application(s)/service(s) and/or their components while adhering to the proper versioning procedure required by TS.
- Assists in installation of the SUT, including all associated databases with appropriate data volume, in the Test Center.
- Responds to system architectural and other technical questions as needed by TS.
- Provides a demonstration of the SUT to TS as the application project is installed in the Test Center.

3.2.2 Testing Service

- Holds Focus Meeting with the Project Team.
- Ensures that key representatives attend all subsequent focused application project calls.
- Defines the scope of each main area of review that must be met for Approval and/or Certification and works with the Project Team regarding the level of engagement necessary to complete the process.
- Determines roles and responsibilities of Testing Service and SUT personnel for all activities.
- Reviews all documentation for the SUT and dependency application(s)/service(s) and/or their components.
- Verifies that the requirements for the appropriate ProPath process reviews have been completed.
- Attends demonstration of the SUT presented by the Project Team using the newly installed application.
- Prepares Test Scripts
    1. Develop IV&V and/or performance test scripts for all testable requirements including anticipated results for each requirement using IV&V Test Director Standards and Conventions as a guideline for process and naming standards.
    2. Determine which test cases and scripts are candidates for automation.

3. Develop, test, and debug automated testing scripts.

- Packages test scripts for execution into test sets in Test Director.

3.3 Conduct IndependentTesting Activities, Initial Build Testing

3.3.1 Project Team

- Collaborates with TS as needed on necessary system and integration testing to verify that all of the repaired components of the application are operational as installed in the provisioned environment. This test must embody all functional system components that make up the overall system and verify that all system and subsystem components operate as anticipated. Subsystem components include any dependency application(s)/ service(s) and/or their components.
- Receives and acts upon test defect(s) reported by TS.
- Coordinates with all escalation processes in conjunction with TS.

3.3.2 Testing Service

- Populates all test databases with test data.

NOTE: *Task complexity is predicated on the existence of migration processes, data support processes, and/or testable data sets.*

1. Develop methods/code/scripts, if necessary, to support the population of databases with the volume necessary to conduct IV&V and performances tests.
2. Populate databases with data.
3. Backup the populated databases, before any testing activity occur, for rollback procedures.

- Executes Independent Verification and Validation (IV&V) and Performance Tests.

1. Run tests by utilizing methodology prescribed in IV&V Test Plan and/or the Performance Test Plan (PTP).
2. Analyze test results.
3. Provide Defect Report for tests that did not meet expected results for IV&V tests.
4. Provide Point Reports during iterative test cycles as necessary for performance related problems.
5. Provide additional results reports as appropriate.
6. Collaborate with the Project Team and dependency project team(s) to resolve both IV&V and performance problems as necessary and/or escalate if build is deemed unviable for testing.

NOTE: *Any builds deemed unviable for testing will be declared as such to the Development Manager and must cycle back through the Testing Service workload schedule calendar.*

- Performs Roll-Out tests

1. Reinstall the SUT (if necessary) with subsequent viable build(s).

NOTE: *It is expected that multiple viable builds will result from the testing efforts.*

2. Evaluate the installation instructions and supporting documentation for the SUT.

3. If failures occur and/or documentation does not support the application project as well as dependency application(s)/service(s) and/or their components, request documentation updates from the Project Team and return to the Prepare Test Scripts sub-activity.

3.4 Conduct Independent Testing Activities, Subsequent Build Testing

3.4.1 Project Team

- Completes new viable application builds without introducing new functionality as needed until the application project is deemed to be acceptable for promotion.
- Provides a Service Request for the testing of the new viable build using the Testing Service Request System (TSRS) available by following this link: http://vista.med.va.gov/testingservice/requests_tsrs.asp
- Delivers all required input artifacts for the new build as defined in the Testing Service Entrance Criteria.
- Provides a detailed list of both functional and design changes introduced into the new build for the SUT as well as dependency application(s)/service(s) and/or their components.

NOTE: *New functionality must* **not** *be introduced into the application project. Any deviation from the scope of the original build test plan requires scheduling changes to the TS schedule calendar.*

- Provides documentation that the SUT has successfully completed the appropriate ProPath processes reviews for the new build.
- Provides guidance and support to TS throughout the Approval and/or Certification process.
- Provides all appropriate builds of the application project as well as dependency application(s)/service(s) and/or their components while adhering to the proper versioning procedure required by TS.

NOTE: *All builds must be viable and ready for testing before entering TS.*

- Assists in the installation of the SUT, including all associated databases with appropriate data volume, in the Test Center.
- Responds to system architectural and other technical questions as needed by TS.
- Collaborates with TS as needed on necessary system and integration testing to verify that all repaired components of the application are operational as installed in the provisioned environment. This test must embody all functional system components that make up the overall system and verify that all system and subsystem components operate as anticipated. Subsystem components include any dependency application(s)/ service(s) and/or their components.
- Receives and acts upon defect(s) reported by TS.
- Coordinates all escalation processes in conjunction with TS.

3.4.2 Testing Service

- Defines the scope of retest of the new build, assuming the SUT has already been through TS and received the necessary approvals for UFT.
- Determines roles and responsibilities of Testing Service and SUT personnel for the retest activities of the new build.
- Reviews all revisions of the documentation for the SUT as well as dependency application(s)/service(s) and/or their components.

- Analyzes the database requirements necessary to install the SUT databases in the Test Center. This includes determination of the system to be used, size, method or technique, and responsibility for populating the all VistA, Cache, Oracle, SQL Server, and /or other databases used by the SUT.

    1. Determine volumes of data necessary to support performance testing for all databases used by the SUT. Use the Package Resource Evaluation (PRE), Package Event Modeling Analysis (PEMA) and Package User Modeling Analysis (PUMA) reports as needed. PRE, PEMA and PUMA reports are defined in Testing Services Capacity Planning Processes.

    2. Determine methods for populating databases with additional data volume to support the performance tests.

- Confirms the system environment requirements necessary for reinstallation of the new build.
- Confirms the approach for retesting activities of the new build.
- Builds the environment utilizing updated installation guides provided by the Project Team for the application project as well as dependency application(s)/service(s) and/or their components.
- Populates all test databases with test data.

NOTE: *Task complexity is predicated on the existence of migration processes, data support processes, and/or testable data sets.*

1. Develop methods/code/scripts, if necessary, to support the population of databases with the volume necessary to conduct IV&V and performances tests.

2. Populate test databases with test data.

3. Backup the populated databases before any testing activity has occurred for rollback procedures.

- Verifies the completion of system and integration testing as accomplished by the Project Team. This activity ensures that all repaired components of the application are operational as installed and that the application functions as expected in the new environment. This is done prior to commencement of the IV&V and performance testing activities as an important step to ensure the SUT is suitable for retesting and to limit testing script issues.
- Prepares additional Test Scripts as needed.

    1. Develop IV&V and/or performance test scripts for all testable requirements, including anticipated results for each requirement using IV&V Test Director Standards and Conventions as a guideline for process and naming standards.

    2. Determine which test cases and scripts are candidates for automation.

    3. Develop, test, and debug automated testing scripts.

    4. Package test scripts for execution into test sets in Test Director.

- Executes Independent Verification and Validation and Performance Tests.

    1. Run tests by utilizing methodology prescribed in IV&V Test Plan and/or the Performance Test Plan (PTP).

    2. Analyze test results.

    3. Provide Defect Report for tests that did not meet expected results for IV&V tests.

    4. Provide Point Reports during iterative test cycles as necessary for performance related problems.

5. Provide additional results reports as appropriate.
6. Collaborate with the Project Team and dependency project team(s) to resolve both IV&V and performance problems as necessary and/or escalate if new build is deemed unviable for testing.

NOTE: *Any builds deemed unviable for testing will be declared as such to the Development Manager. Subsequent testing will be scheduled based on the TS workload schedule calendar.*

- Performs Roll-Out tests

1. Reinstall the SUT (if necessary) with subsequent viable build(s).

NOTE: *It is expected that multiple viable builds will result from the testing efforts.*

2. Evaluate the installation instructions and supporting documentation for the SUT.
3. If failures occur and/or documentation does not lend itself well to the support of the application project and dependency application(s)/service(s) and/or their components, request documentation updates from Project Team and return to the Executes Independent Verification and Validation and Performance Tests sub-activity.

3.5 Grant Testing Service Approval To Proceed

3.5.1 Project Team

- Obtains Approval to Proceed from TS

3.5.2 Testing Service

- Produces Testing Service Approval to Proceed

1. If Testing Service testing is successful, produce correlating test analysis results deliverables.
2. Issue Approval Documents that provide necessary approvals for proper versions of the SUT. The Approval document also outlines the scope of the approval in detail.

- Issues any necessary final reports:

1. Guidance Documents
2. Result Documents
3. Approval and/or Certification Documents

- Catalogues all use-case test scripts that are associated with the final build in TS Script Library (i.e., Test Director) for later use in comparison and load-testing exercises.
- Updates, closes-out, and backs up all files and tracking vehicles associated with the final application project build to permit future historical viewing.
- Uses new information derived from performance testing to validate capacity forecasts and models to improve accuracy in future evaluation exercises.

4 Processes and Related References

4.1 Related Processes

- Product Build
- Test Preparation
- Test and Certification

4.2 Related References

- Testing Service Web page

http://vista.med.va.gov/testingservice/default.asp

- SQA Review Checklist
  http://vaww.oed.wss.va.gov/process/Library/sqa_patch_review_checklist.xls
- ProPath Reviews Guide
  http://vaww.oed.wss.va.gov/process/Library/propath_reviews_guide.doc
- ASSESS Preliminary Form

http://vista.med.va.gov/capman/Assess/Assess%20Forms/ASSESS08_PRELIMINARY_FORM.doc

- ASSESS Final Form

http://vista.med.va.gov/capman/Assess/Assess%20Forms/ASSESS08_FINAL_FORM.doc

- Project Team Kick-off Meeting Guide
  http://vaww.oed.wss.va.gov/process/Library/project_team_kick-off_meeting_guide.doc
- Predictive Capacity Planning Analysis Guide

http://vista.med.va.gov/SEPG_lib/Standard%20Operating%20Procedures/192-034%20Predictive%20Capacity%20Planning%20Analysis%20Standard.doc

- Mandatory Checklist

http://vista.med.va.gov/SEPG_lib/Standard%20Operating%20Procedures/192-034%20Predictive%20Capacity%20Planning%20Analysis%20Standard.htm

APPENDIX E – VistA CPRS Text Integration Utilities (TIU) Generic HL7 Interface Handbook (Test Appendix) Example for Revision

The HL7 Logical Link File #870 – Link for DDC

Logical links describe the complete network path to a given system. The ROES sending application is configured to receive APPLICATION ACKNOWLEDGMENTS (AA) over the same port it is sending HL7 messages, so only one LOGICAL LINK is necessary. Your site should already have a link ready for this use.

- The Logical Link for DDC is VADDC.
- TCP/IP Service Type for the link must be set as "Client (Sender)".
- The IP address should be set to 10.224.54.3.

If DDC determines that a second link is necessary for APPLICATION ACKNOWLEDGMENTS (AA), the DDC POC will contact you regarding the correct name, IP Address and Port # to be used. If you had to edit the TCP/IP Service Type or IP address for this link save the changes and return to the HL7 Main Menu Option.
From the HL7 Main Menu option, select the Filer and Link Management Options menu. To test the link, select "Ping (TCP Only)" and at the prompt "Select HL LOGICAL LINK NODE:" enter VADDC. The message "PING Worked" should be displayed. If you do not see this message, ensure that the receiving system has been configured and setup properly. After the PING test, the link must be started. Here's the menu and option:

```
Select HL7 Main Menu Option: Filer and Link Management Options
SM Systems Link Monitor
FM Monitor, Start, Stop Filers
LM TCP Link Manager Start/Stop
SA Stop All Messaging Background Processes
RA Restart/Start All Links and Filers
DF Default Filers Startup
SL Start/Stop Links
PI Ping (TCP Only)
ED Link Edit
ER Link Errors ...

Select Filer and Link Management Options Option: SL Start/Stop Links

[This option is used to launch the lower level protocol for the appropriate
device. Please select the node with which you want to communicate, in this case,
VADDC.]

Select HL LOGICAL LINK NODE: VADDC
```