

OmniORB 4.1.1 Modifications

Introduction

This document outlines the modifications made to OmniORB version 4.1.1 to support LiveCycle ES (version 8.2.1) components.

Update for Windows wchar_t

Changed the omniORB build script for Windows to build using the -Zc:wchar_t option. This option will make the type wchar_t a native type.

Modified file: OmniORB-4.1.1/ mk/x86_win32_vs_8.mk

Added -Zc:wchar_t to the following flags:

```
MSVC_DLL_CXXNODEBUGFLAGS
MSVC_DLL_CXXDEBUGFLAGS
MSVC_STATICLIB_CXXNODEBUGFLAGS
MSVC_STATICLIB_CXXDEBUGFLAGS
```

Update CA to Verify Client Certificates

Updated CA to verify client certificates.

Modified the file: omniORB-4.1.1/src/lib/omniORB/orbcore/ssl/sslContext.cc

Modified the method:

```
void sslContext::set_CA ()
```

APPLIES TO

LiveCycle ES (version 8.2.1)

CONTENTS

Introduction	1
Update for Windows wchar_t	1
Update CA to Verify Client Certificates.....	1
Added Disable External Settings	2
Added wait_for_work()	3
Support IPv4 in IPv6 in localhost transport rule	6
HPUX returns an invalid protocol from getaddrinfo() with PF_UNSPEC.....	7

After:

```
if (!(SSL_CTX_load_verify_locations(pd_ctx,pd_cafile,0))) {
    report_error();
    OMNIORB_THROW(INITIALIZE,INITIALIZE_TransportError,CORBA::
        COMPLETED_NO);
}
```

Added:

```
// Scan all certificates in CAfile and list them as acceptable CAs
SSL_CTX_set_client_CA_list(pd_ctx,SSL_load_client_CA_file(pd_cafile));

// Define if client certificate is ignored, requested or required
SSL_CTX_set_verify(pd_ctx,SSL_VERIFY_PEER|SSL_VERIFY_FAIL_IF_NO_PEER_CERT
    ,NULL);
```

Added Disable External Settings

Introduced a new command line option `-ORBdisableExternalSettings 0/1` (default 0). When this option is turned on (1), only the command line arguments and the options passed to `ORB_init()` are loaded (skipping environment variables, configuration file and Windows registry).

This helps to prevent from using the external settings as a back door

Modified the file: omniORB-4.1.1/src/lib/omniORB/orbcore/corbaOrb.cc

Added the code snippet after `static void enableLcdMode() {...}`:

```
// Begin code snippet
static CORBA::Boolean disableExternalSettings = (CORBA::Boolean) 0;

class disableExternalSettingsHandler : public orbOptions::Handler {
public:

    disableExternalSettingsHandler() :
        orbOptions::Handler("disableExternalSettings",
            "disableExternalSettings = 0 or 1",
            1,
            "-ORBdisableExternalSettings < 0 | 1 >") {}

    void visit(const char* value,orbOptions::Source) throw
        (orbOptions::BadParam) {
        CORBA::Boolean v;
        if (!orbOptions::getBoolean(value, v)) {
            throw orbOptions::BadParam(key(), value,
                orbOptions::expect_boolean_msg);
        }
    }

    void dump(orbOptions::sequenceString& result) {
        orbOptions::addKVBoolean(key(), disableExternalSettings, result);
    }
}
```

```
};

static disableExternalSettingsHandler disableExternalSettingsHandler_;
// End code snippet
```

Modifications to the function CORBA::ORB_ptr:

```
CORBA::ORB_init(int& argc, char** argv, const char* orb_identifier,
               const char* options[][2])
```

After:

```
orbOptions::singleton().reset();
```

Added:

```
for (int i=argc-1; i>=0; i--) {
    if (!strcmp(argv[i], "-ORBdisableExternalSettings") && (i+1 < argc)) {
        orbOptions::getBoolean(argv[i+1], disableExternalSettings);
        break;
    }
}
```

After:

```
config_fname = orbOptions::singleton().getConfigFileName(argc, argv,
config_fname);
```

Added:

```
if (!disableExternalSettings) {
```

After:

```
orbOptions::singleton().importFromEnv();
```

Add:

```
} // !disableExternalSettings
```

At the end of omni_corbaOrb_initialiser() {...} before the final closing brace } add

```
orbOptions::singleton().registerHandler(disableExternalSettingsHandler_);
```

Added wait_for_work()

Added wait for work to omniORB

Modified the file: omniORB-4.1.1/include/omniORB4/CORBA_ORB.h

Added:

```
virtual Boolean wait_for_work() = 0;
```

After:

```
virtual void run() = 0;
```

Modified the file: omniORB-4.1.1/include/omniORB4/internal/corbaOrb.h

Added:

```
virtual CORBA::Boolean wait_for_work();
```

After:

```
virtual void run();
```

Added:

```
CORBA::Boolean wait_for_work_timeout(unsigned long secs, unsigned long nanosecs);
```

After:

```
CORBA::Boolean run_timeout(unsigned long secs, unsigned long nanosecs);
```

Modified the file: omniORB-4.1.1/include/omniORB4/internal/invoker.h

Added:

```
void wait_for_work(unsigned long secs = 0, unsigned long nanosecs = 0);
```

After:

```
void perform(unsigned long secs = 0, unsigned long nanosecs = 0);
```

Modified the file: omniORB-4.1.1/include/omniORB4/omniAsyncInvoker.h

Added:

```
virtual void wait_for_work(unsigned long secs = 0, unsigned long nanosecs = 0);
```

After:

```
virtual void perform(unsigned long secs = 0, unsigned long nanosecs = 0);
```

Modified the file: omniORB-4.1.1/src/lib/omniORB/orbcore/corbaOrb.cc

Added the code snippet for the following functions:

```
CORBA::Boolean omniOrbORB::wait_for_work()
CORBA::Boolean omniOrbORB::wait_for_work_timeout(unsigned long secs,
unsigned long nanosecs)
```

after the function CORBA::Boolean omniOrbORB::run_timeout(unsigned long secs, unsigned long nanosecs) {...}

```
// Begin code snippet
CORBA::Boolean
omniOrbORB::wait_for_work()
{
    CHECK_NOT_NIL_SHUTDOWN_OR_DESTROYED();

    if (orbAsyncInvoker)
        orbAsyncInvoker->wait_for_work();
}
```

```

    return pd_shutdown;
}

CORBA::Boolean
omniOrbORB::wait_for_work_timeout(unsigned long secs, unsigned long
nanosecs)
{
    CHECK_NOT_NIL_SHUTDOWN_OR_DESTROYED();

    if (orbAsyncInvoker) {
        orbAsyncInvoker->wait_for_work(secs, nanosecs);
    }

    return pd_shutdown;
}
// End code snippet

```

Added the code snippet for the following function:

```

void ORBAsyncInvoker::wait_for_work(unsigned long secs, unsigned long
nanosecs)

```

After the function `int ORBAsyncInvoker::work_pending() {...}`

```

// Begin code snippet
void ORBAsyncInvoker::wait_for_work(unsigned long secs, unsigned long
nanosecs)
{
    orb_lock.lock();
    invoker_threads++;

    while (!invoker_shutting_down &&
omniTaskLink::is_empty(invoker_dedicated_tq)) {

        // Wait for a task to arrive
        if (secs || nanosecs) {
            if (invoker_signal.timedwait(secs, nanosecs) == 0) {
                // timeout
                invoker_threads--;
                if (invoker_shutting_down) invoker_signal.signal();
                orb_lock.unlock();
                return;
            }
        }
        else {
            invoker_signal.wait();
        }
    }

    invoker_threads--;
    if (invoker_shutting_down) invoker_signal.signal();
    orb_lock.unlock();
}
// End code snippet

```

Modified the file: omniORB-4.1.1/src/lib/omniORB/orbcore/invoker.cc

Added the code snippet for the following function:

```
void omniAsyncInvoker::wait_for_work(unsigned long secs, unsigned long nanosecs)
```

after the function `void omniAsyncInvoker::perform(unsigned long secs, unsigned long nanosecs) {...}`

```
// Begin code snippet
```

```
void
omniAsyncInvoker::wait_for_work(unsigned long secs, unsigned long
                                nanosecs)
{
    omniORB::logs(1, "omniAsyncInvoker::wait_for_work() not implemented.
                    aborting...\n");
    abort();
}
// End code snippet
```

Support IPv4 in IPv6 in localhost transport rule

Ported revision 1.1.4.8 changes from OmniORB head into OmniORB 4.1.1 to properly support IPv4 in IPv6 in localhost transport rule.

Modified the file: omniORB-4.1.1/src/lib/omniORB/orbcore/transportRules.cc

Modified the method

```
static char* extractHost(const char* endpoint)
```

Replaced the code under:

```
CORBA::String_var host = omniURI::extractHostPort(p, port, 0);
```

With:

```
if (LibcWrapper::isip4addr(host)) {
    return host._retn();
}
else if (LibcWrapper::isip6addr(host)) {
    // Check if it's IPv4 encapsulated in IPv6
    if (strncasecmp(host, "::ffff:", 7) == 0 &&
        LibcWrapper::isip4addr((const char*)host + 7)) {

return CORBA::string_dup((const char*)host + 7);
    }
    return host._retn();
}
else {
    // Try to resolve name
    LibcWrapper::AddrInfo_var ai(LibcWrapper::getAddrInfo(host, port));
    if (ai.in())
        return ai->asString();
}
```

```

    }
}
return 0;

```

HPUX returns an invalid protocol from getaddrinfo() with PF_UNSPEC.

Ported revision 1.1.4.17 changes from OmniORB head into OmniORB 4.1.1 to fix an HPUX issue so that it retries IPv4 protocol if IPv6 does not work.

Modified the file: omniORB-4.1.1/src/lib/omniORB/orbcore/tcp/tcpEndpoint.cc

Modified the method

```

CORBA::Boolean
tcpEndpoint::Bind() {

```

Replaced the entire method with the following code:

```

CORBA::Boolean
tcpEndpoint::Bind() {

    OMNIORB_ASSERT(pd_socket == RC_INVALID_SOCKET);

    const char* host;
    int passive_host; // 0 for explicit, 1 for unspecified passive, 2
                      // for IPv4 passive, 3 for IPv6 passive.

    if ((char*)pd_address.host && strlen(pd_address.host) != 0) {
        host = pd_address.host;

        if (omni::strMatch(pd_address.host, "0.0.0.0")) {
            passive_host = 2;
        }
#ifdef OMNI_SUPPORT_IPV6
        else if (omni::strMatch(pd_address.host, ":::")) {
            passive_host = 3;
        }
#endif
        else {
            if (omniORB::trace(25)) {
                omniORB::logger log;
                log << "Explicit bind to host " << pd_address.host << ".\n";
            }
            passive_host = 0;
        }
    }
    else {
#ifdef OMNI_IPV6_SOCKETS_ACCEPT_IPV4_CONNECTIONS ||
defined(IPV6_V6ONLY)
        host = 0;
        passive_host = 1;
#else
        host = "0.0.0.0";

```

```

    passive_host = 2;
#endif
}

LibcWrapper::AddrInfo_var ai;

do {
    ai = LibcWrapper::getAddrInfo(host, pd_address.port);

    if ((LibcWrapper::AddrInfo*)ai == 0) {
        if (omniORB::trace(1)) {
omniORB::logger log;
log << "Cannot get the address of host " << host << ".\n";
        }
        CLOSESOCKET(pd_socket);
        return 0;
    }

    pd_socket = socket(ai->addrFamily(), SOCK_STREAM, 0);

    if (pd_socket == RC_INVALID_SOCKET) {

        if (passive_host == 1) {
omniORB::logs(2, "Unable to open socket for unspecified passive host "
                " -- fall back to IPv4.");
            host = "0.0.0.0";
            passive_host = 2;
            continue;
        }
        else {
omniORB::logs(1, "Unable to open required socket.");
            return 0;
        }
    }

#if defined(OMNI_SUPPORT_IPV6) && defined(IPV6_V6ONLY)
# if !defined(OMNI_IPV6_SOCKETS_ACCEPT_IPV4_CONNECTIONS)
    if (passive_host == 1) {
        // Attempt to turn IPV6_V6ONLY option off
        int valfalse = 0;
        omniORB::logs(10, "Attempt to set socket to listen on IPv4 and IPv6.");

        if (setsockopt(pd_socket, IPPROTO_IPV6, IPV6_V6ONLY,
            (char*)&valfalse, sizeof(valfalse)) == RC_SOCKET_ERROR) {
omniORB::logs(2, "Unable to set socket to listen on IPv4 and IPv6. "
                "Fall back to just IPv4.");
CLOSESOCKET(pd_socket);
pd_socket = RC_INVALID_SOCKET;
host = "0.0.0.0";
passive_host = 2;
        }
    }
# endif
#endif
}
#endif
#endif

```

```

} while (pd_socket == RC_INVALID_SOCKET);

{
    // Prevent Nagle's algorithm
    int valtrue = 1;
    if (setsockopt(pd_socket, IPPROTO_TCP, TCP_NODELAY,
        (char*)&valtrue, sizeof(int)) == RC_SOCKET_ERROR) {

        omniORB::logs(2, "Warning: failed to set TCP_NODELAY option.");
    }
}

if (orbParameters::socketSendBuffer != -1) {
    // Set the send buffer size
    int bufsize = orbParameters::socketSendBuffer;
    if (setsockopt(pd_socket, SOL_SOCKET, SO_SNDBUF,
        (char*)&bufsize, sizeof(bufsize)) == RC_SOCKET_ERROR) {
        CLOSESOCKET(pd_socket);
        pd_socket = RC_INVALID_SOCKET;
        omniORB::logs(1, "Failed to set SO_SNDBUF option.");
        return 0;
    }
}

SocketSetCloseOnExec(pd_socket);

if (pd_address.port) {
    int valtrue = 1;
    if (setsockopt(pd_socket, SOL_SOCKET, SO_REUSEADDR,
        (char*)&valtrue, sizeof(int)) == RC_SOCKET_ERROR) {

        omniORB::logs(2, "Warning: failed to set SO_REUSEADDR option.");
    }
}

if (omniORB::trace(25)) {
    omniORB::logger log;
    CORBA::String_var addr(ai->asString());
    log << "Bind to address " << addr << " ";
    if (pd_address.port)
        log << "port " << pd_address.port << ".\n";
    else
        log << "ephemeral port.\n";
}

if (::bind(pd_socket, ai->addr(), ai->addrSize()) == RC_SOCKET_ERROR) {
    if (omniORB::trace(1)) {
        omniORB::logger log;
        CORBA::String_var addr(ai->asString());
        log << "Failed to bind to address " << addr << " ";
        if (pd_address.port)
            log << "port " << pd_address.port << ". Address in use?\n";
        else
            log << "ephemeral port.\n";
    }
    CLOSESOCKET(pd_socket);
}

```

```

    return 0;
}

if (listen(pd_socket, SOMAXCONN) == RC_SOCKET_ERROR) {
    CLOSESOCKET(pd_socket);
    omniORB::logs(1, "Failed to listen on socket.");
    return 0;
}

// Now figure out the details to put in IORs

OMNI_SOCKADDR_STORAGE addr;
SOCKNAME_SIZE_T l;
l = sizeof(OMNI_SOCKADDR_STORAGE);
if (getsockname(pd_socket,
    (struct sockaddr *)&addr, &l) == RC_SOCKET_ERROR) {
    CLOSESOCKET(pd_socket);
    omniORB::logs(1, "Failed to get socket name.");
    return 0;
}
pd_address.port = tcpConnection::addrToPort((struct sockaddr *)&addr);

if (passive_host) {
    // Ask the TCP transport for its list of interface addresses

    CORBA::ULong  addr_len = 0;
    CORBA::Boolean set_host = 0;

    const omnivector<const char*>* ifaddrs
        = giopTransportImpl::getInterfaceAddress("giop:tcp");

    if (ifaddrs && !ifaddrs->empty()) {
        // TCP transport successfully gave us a list of interface addresses

        const char* loopback4 = 0;
        const char* loopback6 = 0;

        omnivector<const char*>::const_iterator i;
        for (i = ifaddrs->begin(); i != ifaddrs->end(); i++) {

            if (passive_host == 2 && !LibcWrapper::isip4addr(*i))
                continue;

            if (passive_host == 3 && !LibcWrapper::isip6addr(*i))
                continue;

            if (omni::strMatch(*i, "127.0.0.1")) {
                loopback4 = *i;
                continue;
            }
            if (omni::strMatch(*i, ":::1")) {
                loopback6 = *i;
                continue;
            }
        }
    }
}

```

```

pd_addresses.length(addr_len + 1);
pd_addresses[addr_len++] = omniURI::buildURI("giop:tcp:",
        *i, pd_address.port);

if (!set_host) {
    pd_address.host = CORBA::string_dup(*i);
    set_host = 1;
}
}
    if (!set_host) {
// No suitable addresses other than the loopback.
if (loopback4) {
    pd_addresses.length(addr_len + 1);
    pd_addresses[addr_len++] = omniURI::buildURI("giop:tcp:",
        loopback4,
        pd_address.port);
    pd_address.host = CORBA::string_dup(loopback4);
    set_host = 1;
}
if (loopback6) {
    pd_addresses.length(addr_len + 1);
    pd_addresses[addr_len++] = omniURI::buildURI("giop:tcp:",
        loopback6,
        pd_address.port);
    if (!set_host) {
        pd_address.host = CORBA::string_dup(loopback6);
        set_host = 1;
    }
}
}
if (!set_host) {
    omniORB::logs(1, "No suitable address in the list of "
        "interface addresses.");
    CLOSESOCKET(pd_socket);
    return 0;
}
}
}
else {
    omniORB::logs(5, "No list of interface addresses; fall back to "
        "system hostname.");
    char self[OMNIORB_HOSTNAME_MAX];

    if (gethostname(&self[0], OMNIORB_HOSTNAME_MAX) == RC_SOCKET_ERROR) {
omniORB::logs(1, "Cannot get the name of this host.");
CLOSESOCKET(pd_socket);
return 0;
    }
    if (orbParameters::dumpConfiguration || omniORB::trace(10)) {
omniORB::logger log;
log << "My hostname is '" << self << "'.\n";
    }
    LibcWrapper::AddrInfo_var ai;
    ai = LibcWrapper::getAddrInfo(self, pd_address.port);
    if ((LibcWrapper::AddrInfo*)ai == 0) {

```

```

if (omniORB::trace(1)) {
    omniORB::logger log;
    log << "Cannot get the address of my hostname '"
        << self << "'.\n";
}
CLOSESOCKET(pd_socket);
return 0;
}
pd_address.host = ai->asString();
pd_addresses.length(1);
pd_addresses[0] = omniURI::buildURI("giop:tcp:",
    pd_address.host, pd_address.port);
}
if (omniORB::trace(1) &&
(omni::strMatch(pd_address.host, "127.0.0.1") ||
omni::strMatch(pd_address.host, ":::1"))) {

    omniORB::logger log;
    log << "Warning: the local loop back interface (" << pd_address.host
    << ")\n"
    << "is the only address available for this server.\n";
}
}
else {
    // Specific host
    pd_addresses.length(1);
    pd_addresses[0] = omniURI::buildURI("giop:tcp:",
        pd_address.host, pd_address.port);
}

// Never block in accept
SocketSetnonblocking(pd_socket);

// Add the socket to our SocketCollection.
addSocket(this);

return 1;
}

```