# NetReady™ 3.0 Device Agent Integration Requirements

**Document Version 3.0.0**

UpdateLogic Incorporated

508-624-8688 (TEL)

508-624-8686 (FAX)

http://www.updatelogic.com

Thank you for using the UpdateLogic NetReady™ 3.0. NetReady provides comprehensive, cost-effective services which allow remote support sessions, software and firmware updates, secure field provisioning of DRM keys, and complete device management for all types of Internet-enabled CE devices.

If you find any problems with this software, please report them via email to support@updatelogic.com.

Please visit our website at www.updatelogic.com for the latest news and information.

UpdateLogic Incorporated

508-624-8688 (TEL)

508-624-8686 (FAX)

# Table of Contents

# 1 Overview

For over 50 years, the television industry has been operating on analog technology. Analog televisions contain little or no computational abilities, and have therefore been sold and marketed much the same as a dishwasher or any other home appliance. With the advent of high definition digital content, low-cost digital circuitry that supports Internet connections, and the ubiquity of streaming media services, the digital age is reaching the most powerful of all home appliances, the device.

The connected devices such as TVs and Blu-ray players will soon compete with the home computer as the most complex device in the home. Software updates, the provisioning of streaming media security credentials, remote support facilities are now all a part of modern connected CE devices. UpdateLogic's NetReady technology provides all of these services.

## 1.1 Change Tracking

| **Version Number of This Document** | **Changes** |
|---|---|
| 2.0.0 | Original version for 2.0 |
| 2.0.1 | Broke out NetProvision vs. NetUpdate requirements. |
| 3.0.0 | Updated to latest NetReady and SMS terms. |

## 1.2 Scope

This document describes the platform, factory, and development requirements for devices supporting the NetReady Device Agent. The Device Agent is the part of the NetReady solution that resides on a device and manages all update, provisioning, diagnostic data collection, and remote support functionality by conversing with the NetReady SMS.

## 1.3 Audience

This document is designed to be read by all parties interested in the NetReady solution including management, technical, compliance, quality assurance, and marketing personnel. Readers should be well versed in connected digital television technology including internet protocols, digital receiver architecture, MPEG protocols, and embedded software development.

## 1.4 Definitions

This section provides a glossary of terms used in this document.

| **Term** | **Definition** |
|---|---|
| Device Agent | The UpdateLogic software that resides on a device and enables it to talk to the NetReady SMS and performs update, provisioning, and diagnostic functions. |
| CEM | Consumer Electronics Manufacturer |
| CSP | Content Service Provider |
| NOC | The UpdateLogic Network Operating Center – a system of hardware and software that resides |

| | |
|---|---|
| | in the Internet cloud that a device contacts to receive its keys. |
| OAD | Over the Air Download |
| SoC | System on Chip |
| ULI | UpdateLogic, Inc. |
| ULPK | UpdateLogic Provisioning Key.  A device-unique key that is inserted at the factory to authenticate a device when it registers with the NOC. |

## 2 Introduction

NetReady is a suite of technologies that enable a CE manufacturer to run remote support sessions, update the software, data, and security credentials of a device after it ships, store registration information, and acquire real-time diagnostic information to help troubleshoot problems.  Each of these technologies is independent of the other.  In order to use one you do not have to include the others.  This section provides a brief synopsis for each of these technologies.

Any connected device can be enabled with the Device Agent to support the NetReady suite of services.  The Device Agent is a small, standards-based program that is integrated into an Internet connected device by the OEM.  Each of the three main NetReady services can be optionally enabled or disabled at compile time.  The Device Agent is responsible for communicating with the NetReady SMS through which it starts live support sessions, detects and downloads firmware updates, and receives provisioned objects.  The Device Agent provides device authentication, error correction, decryption, and image authentication.

## 3 Platform Requirements

The Device Agent is written in ANSI C and has been ported to dozens of platforms including variations in CPU type and speed, memory size and speed, middleware, and operating system.  The Device Agent was design to support such platform features as power management, user abort, and dual tuners.  To expedite integration, the Device Agent source code is partitioned into two sections: core and platform adaptation.  The core software contains code and data that is common to all implementations of the Device Agent.  This core code maintains compatibility with the NetReady SMS and should not be modified during integration.  The second section is the platform adaptation layer.  The platform adaptation layer is ported by the OEM to the target platform.  The resulting software contains all the platform specific code including operating system, memory, Internet interface, SSL, tuner interface, and section filter interface.  The isolation of the core code from the platform adaptation layer simplifies the integration process by insulating the developer from UpdateLogic network protocol specifics.

Complete information regarding the sources and how to port them may be found in the *NetReady 3.0 Device Agent Integration Instructions.* The source code is fully commented and is the ultimate reference.

## 3.1 General Platform Requirements

This section describes requirements that are common to both connected internet-based devices and unconnected devices that use over-the-air downloads for updates.

### 3.1.1 Operating System

The Device Agent software makes use of POSIX-style APIs for OS-specific features such as threads, timers, memory management, and file access. The Device Agent is shipped with an example interface for Linux. If your platform's OS is Linux then in most cases the example Linux interface can be used as is. If your platform's OS is not Linux then you can use the example Linux interface as a template to build out the Device Agent's OS-specific interfaces for your target's OS. The Device Agent may be configured to run in its own thread in a multi-threaded environment or as an application that runs with controlled yields in a single-threaded environment.

### 3.1.2 Binary Size

The core agent typically compiles into binary that is ~150 KB in size.

Enabling all live support functionality including streaming video typically adds ~250 KB to the size of the agent.

### 3.1.3 Memory Usage

The core agent uses ~500 KB of dynamic memory.

Use of software updates uses a configurable amount of dynamic memory that is typically ~2 MB.

Use of live support without streaming video typically adds ~500 KB of dynamic memory usage.

Live support streaming video typically adds ~600 KB of dynamic memory usage plus a configurable amount of memory for frame buffering. The amount of dynamic memory used for frame buffering will depend upon the selected frame size and rate. For example, a 960x1080 frame with 4 bits-per-pixel may use ~4 MB of dynamic memory at a slow frame rate such as 2 frames-per-second.

### 3.1.4 NVRAM Usage

The core agent uses about 10K bytes of NVRAM to store persistent data.

Please see the section below if your platform is connected to the Internet and will be using NetReady's device authentication features see section 3.2.3.

### 3.1.5 Library Dependencies

The core agent is not explicitly dependent upon any external libraries.

The complete list of library dependencies when enabling live support is shown below. Note that this includes live support and all optional live support functionality (streaming video, etc…). Many of the libraries will already be present on a device and therefore do not represent additional libraries for the platform due to live support functionality. However, some libraries additions will be necessary for live support. One must cross-reference this list to an existing device platform to identify the new library dependencies due to live support. The size of these libraries will vary based on platform and build options; approximate library sizes are provided for reference.

| Library | ~Size | Library | ~Size |
|---|---|---|---|
| libgio | 1039 KB | libglib | 1200 KB |
| libgmodule | 15 KB | libgobject | 380 KB |
| libgstapp | 82 KB | libgstbase | 330 KB |
| libgstcheck | 79 KB | libgstcontroller | 225 KB |
| libgstcoreelements | 283 KB | libgstdataprotocol | 23 KB |
| libgstffmpegcolorspace | 440 KB | libgstjpeg | 127 KB |
| libgstnet | 44 KB | libgstreamer | 1016 KB |
| libgsttcp | 164 KB | libgstvideo | 62 KB |
| libgstvideoscale | 123 KB | libgthread | 21 KB |
| libintl | 48 KB | libxml2 | 605 KB |
| libz | 121 KB | liboil | 529 KB |
| libm | 149 KB | uuid | 14 KB |

### 3.1.6 Flash Size

When designing the update architecture of a platform there are a few different parameters to consider when designing the size of flash memory to be used in the device. This section describes these considerations.

#### 3.1.6.1 Update Set Considerations

OEM's typically employ multiple "update sets". Sometimes this is called the "ping pong" buffer approach. Multiple update sets allow a platform to receive an update into an area of flash that isn't currently active so that if the update fails for any reason before the final commit then the active set will not be affected. Alternatively, an OEM may decide to only employ a single update set to save on the flash memory cost for additional update sets. This means that if the update fails during the flash write the device may become inoperative. Typically single update sets are only employed on devices with very small updates sets that are assembled in their entirety in memory prior to being committed to flash. Depending on the size of the update and speed of the flash write this approach can bring the risk of failure down to an acceptable level.

A typically connected devices have a large update set. It is therefore impossible to assemble these updates in their entirety in memory before writing them to flash, or their write time is long enough to run a significant risk of failure. Therefore multiple update sets are required for connected devices.

An update set may consist of any combination of partitions, files, or platform-specific objects. If the total size of the update set is 32MB, for example, and two update sets are used then flash memory would need to be 64MB in size.

It’s possible to divide a platform’s software and data into updateable and non-updateable parts. As an example it might be known that certain data files containing a splash screen for example will not be updated. Therefore that part of flash memory would not have to be duplicated to make room for a possible update of that software or data.

It is also possible to embed the Device Agent into a bootloader which is never updated so that the device can go into a “limp home” mode if an update fails due to AC power loss at a time when it is susceptible to flash corruption.

#### 3.1.6.2 MLC Flash Considerations

MLC flash endurance must also be taken into account when considering the total amount of flash required for a device. As MLC flash capacity per dollar increases its bit error rate increases and lifetime erase cycles decrease dramatically. Runtime flash failures should be expected and therefore redundant copies of software and data should be maintained. Total flash size should consider the wear endurance of the flash memory that is chosen for the platform. If flash wear endurance is expected to be exceeded during the normal lifetime of the device then algorithms to detect these failures and switch to backup update sets need to be employed.

### 3.1.7 CPU Speed

***Greater than 200MHz (does not apply to NetReady device authentication functionality)***

While CPU speed in general is not a specific requirement; sufficient processing power is required to decrypt update modules at a reasonable rate of about 2M bytes every minute. If decrypt time falls below this rate then the device can take a long time to update and which may create consumer use case problems, but in and of itself is not a problem. Whenever possible hardware assisted decryption should be used which eliminates any CPU-caused decrypt bottlenecks.

## 3.2 Connected Platform Requirements

This section describes additional requirements that are special to platforms that are connected to the internet and support content service provider (CSP) clients to provide streaming media and other services.

### 3.2.1 Network Connection

***Required***

All connected devices require a physical network connection to the internet such as Ethernet or Wi-Fi that must be supported by associated DHCP client software, a TCP/IP stack, and sockets.

### 3.2.2 Secure Sockets Layer

***Required***

The Device Agent uses SSL to converse with NetReady SMS. The interface to the SSL stack is part of the platform adaptation layer. The example SSL adaptation interface shipped with the Device Agent source uses OpenSSL function calls. If your platform uses OpenSSL then the Agent’s SSL needs will most likely just drop in.

### 3.2.3 NetReady Device Authentication NVRAM Usage

***10K bytes Per CSP***

If the connected Device Agent supports NetReady device authentication functionality then the amount of NVRAM encumbered depends on the number and size of the keys that are managed. Typically each CSP will have keys that are about 10K in size. Therefore if a platform supports three CSPs the target will need to allocate about 30K bytes for CSP keys.

Please see the *NetReady Technical Guide* for more information about NetReady device authentication NVRAM footprint.

## 3.2.4 Secure Boot

***Required***

The software running on a connected device supported by NetReady needs to be authenticated before it is run. Without this authentication platforms can be "opened" which can lead to left of service due to stolen security keys. The sections below outline the requirements for authenticating the platform's software.

#### 3.2.4.1 Signed Boot Code

***Required***

NetReady requires that the CPU checks a hash in the boot code that is encrypted with a key that is CEM-specific and built into the hardware. If this hash check fails then the CPU will not run the boot code.

#### 3.2.4.2 Signed Kernel

***Required***

Just as the CPU checks the boot code, the boot code must check the signed hash of the kernel to authenticate it. If this hash check fails then the boot code will not run the kernel. This is the next step in the security chain that prevents malicious attempts to "open" a device so that its security keys can be stolen.

#### 3.2.4.3 Signed Root File System

***Required***

The final step in platform software security is that the kernel must authenticate the root file system. The most robust way to do this is to sign the entire root file system. Although DigSig keeps rogue executables and libraries from running it doesn't keep rogue shell scripts or attacks based on file replacement, etc. from compromising the security of the platform. If this hash check fails then the kernel must not load the root file system and should halt.

## 3.2.5 Hardware Protected Keys and Cipher Engine

***Required***

At least one and ideally two hardware-protected device-common keys are required to be supported by the SoC. These keys must be part of a security block separate from the main data bus where an AES CBC cipher can process data in a protected fashion. One of the keys and its associated cipher is used to protect data that must be written to/read from flash. This may be accomplished with an encrypted file system as long as the root of trust key is hardware-protected. The second key is used to encrypt the device-unique UpdateLogic Provisioning Keys (ULPKs) for transit to the factory before they're inserted into the devices. This key needs to be pre-shared with UpdateLogic.

### 3.2.6 Electronic Serial Number

***Required***

Each device that the Device Agent is integrated into must be able to return a device-unique serial number that is software readable. This serial number does not need to comply with any format restrictions. Typically whatever format the OEM already uses is adequate. If the OEM does not insert a serial number in the factory the device's MAC address may be used to construct a serial number.

### 3.2.7 Factory Key Insertion

***Required***

Any connected platform that uses NetReady for field provisioning of CSP keys will have to have a device-unique UpdateLogic Provisioning Key (ULPK) inserted at the factory that is used to authenticate the device during registration. The size of this key is dependent on the platform-dependent encryption header used, but is less than 100 bytes. The ULPK can be inserted at the factory in the same manner as device-unique serial numbers, MAC addresses, etc.

### 3.2.8 Real-Time Clock

***Optional, but useful***

Connected devices can take advantage of a real-time clock to throttle the frequency at which they contact NetReady SMS. Systems may, however, be implemented without an RTC.

## 3.3 Over-The-Air Download Requirements

Platforms that include the Device Agent's OAD feature to retrieve updates from the UpdateLogic Nationwide Broadcast network have additional requirements. These requirements are described in this section.

### 3.3.1 Tuner Interface

***Required***

The platform must support an interface to tune the receiver to a given frequency and modulation.

### 3.3.2 Section Filter Interface

***Required***

The platform must support a section filter interface to both set the filter and retrieve the section data that contains the update.

### 3.3.3 External Timer

***Optional, but allows the device to pull less power while waiting for updates***

Because OAD updates are available on a carousel that revolves on a pre-determined schedule a device must be able to sleep for a period of time and wake up to receive an update. If the platform supports an external timer it can go into a low power standby state while it waits for the update to arrive. Of course the platform can simply sleep without the use of an external timer in a fully powered up state while waiting for updates.

# 4 Development Requirements

The Device Agent is distributed as a .TAR file containing both ANSI “C” source and platform-specific private binary libraries. The private library is created by UpdateLogic using an OEM’s tool chain. The .TAR file delivered to an OEM is therefore customized to their platform. This .TAR file is typically unarchived onto an x86 Linux cross-development system where an x86-based reference version of the Device Agent can be quickly built via the included Makefile. This Makefile can be customized to use the target platform’s tool chain or, more typically, the Device Agent source and private library can be included in the target platform’s build environment.

The initial installation of the Device Agent source is less than 20 MB, however, the creation of updates and test transport streams will quickly add to this. Therefore, the disk storage requirements will vary based on the number and size of the updates, carousels, test data, and transport streams you generate. We suggest a minimum of 10 GB free disk space on your cross-development system to store the Device Agent source, examples, and generated updates and associated test transport streams.

# 5 Tools

A set of tools are provided that are used to create updates and define, create, and analyze UpdateLogic DSM-CC A/97 Carousel transport streams. Some of these tools run under both Linux and Windows. Some only run under Windows.

### 5.1.1 Publisher

Publisher is a command line tool that can be run under Linux or Windows. It encrypts and packages a platform’s update components into a format that is digestible by NetReady SMS. For large updates (>32MB) it is recommended that a fast PC with a CPU clock speed of at least 3GHz be used to minimize Publishing time.

Please see the *NetReady Publisher* document for more information.

### 5.1.2 TestCarouselBuilder

TestCarouselBuilder (TCB) is a Windows GUI based tool that creates test carousel transport streams from test template files, Publisher packaged software updates contained in .CAB files, and test data designed to simulate live-network conditions. The templates are used to create test cases designed to expose one or more integration deficiencies.

The transport streams generated by TCB can be played back via a standard playback system such as a Sencore HDTV996A**.**

Please see the *NetReady TestCarouselBuilder* document for more information.

### 5.1.3 TSFactory

TSFactory is a Windows GUI application that is used to create transport streams that can be streamed to receivers on the factory floor to perform a final software update prior to shipment. TSFactory takes from one to four update images as input and outputs a single TS multiplex containing the updates.

Please see the *NetReady TSFactory* documentation for more information.

### 5.1.4 StreamViewer

StreamViewer.exe is a Windows GUI application that is used to analyze TCB and TSFactory generated transport streams containing ULI update carousels.  If the Windows PC that StreamViewer is installed on is equipped with the appropriate DTV tuner card StreamViewer is also capable of monitoring live broadcast streams.

See *NetReady StreamViewer* for more information.