NetReady™ 3.0 Device Agent Integration Instructions

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

DEVICE AGENT PORTING INSTRUCTIONS

This document describes how to port the UpdateLogic Device Agent ("Agent") source code to an internet connected device. An overview of the organization of the source code is provided as well as explicit step-by-step instructions describing how to perform the port. The Agent source comes with example ports which are explained. Various porting issues are discussed at the end of the document.

The Agent supports three different facilities: UpdateTV, SecureTV, and SupportTV. If your target platform doesn't use all three of these facilities you will want to pay attention to notes in this document that indicate whether a particular porting step is special to any one of them. Sections and functions that are for UpdateTV only are marked in red. Sections and functions that are for SupportTV only are marked in blue. There are sections at the end of the document that address special considerations for each of the three different facilities.

RELATED TOOLS RELEASE

This release of the Agent is paired with the release 2.0.1 of the UpdateLogic tools. Please contact UpdateLogic if you need a copy of the Windows self-extracting executable that contains these tools.

CHANGE TRACKING

Version Number of This Document	**Changes**
3.0.0	First release of this document to incorporate SupportTV integration steps.
2.0.26c	In correct function reference in UtvPlatformSecureOpen() description.
2.0.26b	SecureTV only porting details added.
2.0.26	Removal of UtvPlatformSecureSave() and UtvPlatformSecureDelete(). Added UtvPlatformSecureGetULPK().
2.0.24	Consolidated from multiple sources

SCOPE

This document assumes that the reader is familiar with the UpdateLogic network, the Device Agent, and SDK tools. If you are not familiar with these items then please see the appropriate documentation as described in the *Related Documents* section below. Readers should be well versed in digital television technology including digital receiver architecture, internet and MPEG protocols (if they're using OTA updates), and embedded software development.

Important: All developers who are porting the Agent to their receiver should at least read this document as well as the *UpdateLogic Agent Validation Guide.* You will also need to use the tools and so should be familiar with the *NetReady Publisher, NetReady TestCarouselBuilder,* and *NetReady StreamViewer* documents.

Note: Each release of this document is coordinated with a specific Agent source release. Make sure you are working with the appropriate Agent source release.

AUDIENCE

This document is designed to be read by the software engineers who will be incorporating the UpdateLogic Device Agent into a device's application software. It may be useful for engineering management to skim this document to get a sense of what's involved in a port.

RELATED DOCUMENTS

Document	Topic	Source
NetReady Technical Guide	Description of NetReady function and connected device integration. Useful to all audiences.	ULI
NetReady Intro	General introduction to the UpdateLogic technology. Useful to all audiences.	ULI
NetReady Device Agent Integration Requirements	Target software and hardware requirements for your NetReady device.	ULI
NetReady Device Agent Integration Instructions	Step by step porting instructions for the Agent. Useful to engineers porting the UpdateTV, SupportTV, or SecureTV facilities to their device.	ULI
NetReady Device Agent Validation Guide	Description of tests the OEM should run to validate their port of the Agent to a connected device. Useful to all audiences.	ULI
NetReady Factory Process Guide	Introduces NetReady technology to your factory staff prior to production	ULI
NetReady Technical Support Guide	Introduces NetReady technology to your technical support staff prior to production	ULI
UpdateLogic NOC User's Guide	User's manual for interacting with the Network Operating Center via its web interface. Useful for engineers relating to any of the 3 main UpdateLogic facilities.	ULI
NetReady Publisher	User's manual for the Publisher utility. Useful to all audiences.	ULI
NetReady TestCarouselBuilder	User's manual for the TestCarouselBuilder utility. Useful to engineering audiences creating UpdateTV OAD tests.	ULI
NetReady TSFactory	User's manual for the TSFactory utility. Useful to technology evaluators and engineers supporting factory updates using UpdateTV OAD technology.	ULI
NetReady StreamViewer	User's manual for the StreamViewer utility. Useful to engineering audiences working with UpdateTV OAD technology.	ULI
NetReady Factory Update Guide	A document that explains the use of the factory mode of UpdateTV from an operational point of view.	ULI

GLOSSARY OF TERMS

Term	Description
API	Application Programming Interface.
ASI	Asynchronous Serial Interface, often used to distribute MPEG transport streams.
ATSC	Advanced Television Standards Committee. www.atsc.org
Back End	MPEG decode, graphics and user interaction portion of the digital receiver hardware.
BEM	Broadcast Encoder Monitor. A 1U rack mount device that accepts ASI or SMPTE DTV MPEG data in and inserts the UpdateLogic broadcast carousel. Also contains an RF input to monitor the resulting output. This device is installed at both broadcast sites and in customer labs.
CableLabs	Research and development consortium that is dedicated to helping its cable operator members integrate new cable telecommunications technologies. www.cablelabs.com
CAB File	An archive file like a ZIP file that contains one or more files in compressed format.
CEM	Consumer Electronics Manufacturer
CHM	Customer's Hardware Model
CJSON	Third party package providing the implementation of a JSON parser. This package is used by live support functionality of the Agent and is only required if PLATFORM_LIVE_SUPPORT is defined in the Makefile. See ThirdParty/cjson/33/README for additional information.
Component	A separately updateable element within an update.
CMG	Customer's Model Group
Content Service Provider	Content Service Provider. For instance, Netflix, Vudu, Amazon, CinemaNow, etc.
COUI	Customer's Organizational Unique Identifier
CSP	Content Service Provider.
CTV	Connected TV. A TV that supports an Ethernet and/or WiFi connection to the internet and uses that connection to connect to services like streaming media, news, weather, etc..

DCR	Digital Cable Ready. This acronym refers to consumer electronic devices equipped with Digital Tuners and Cable Card slots for use in receiving one-way Digital Cable signals and features.
Front End	Tuner and demodulation portion of the digital receiver hardware.
GStreamer	Third party package used to implement the video capabilities of live support. See http://www.gstreamer.net/ for additional information.
Hardware Model	A number describing a sub-set of a model group that is usually panel size specific.
HM	Hardware Model
HMR	Hardware Model Range
Image	An entire software or data entity that is broken into pieces called “modules” when broadcast on the network.
JSON	Java Script Object Notation (JSON) is a lightweight data-interchange format. JSON is used in the live support capabilities of the Agent. See http://www.json.org/ for additional information.
Manufacturers’ ESN	A serial number that is readable by software and also printed on a label affixed to the outside of a TV. Used by the consumer to identify their TV to technical support staff. Each CEM has a different serial number scheme. The customer can display the serial number by using the TV’s menus. Also called “the SN”.
Manufacturer’s Software Version	A string that contains version information like “TV_System_3.7”. The customer can display the software version using the TV’s menus.
Model Group	A number that uniquely defines a particular TV model within a give OUI (company). A model group may be sub-divided into different hardware models.
MG	Model Group
Module	Name of the distribution unit that a software or data “image” is broken down into in order to broadcast it efficiently on the network.
MV	Module Version
NOC	The UpdateLogic Network Operations Center. A server farm that distributes updates over the Internet and broadcast networks.

NOC Interface	A web-based application that is used to manage updates and provisioning over the UpdateLogic Network. The URL for this application is extdev.updatelogic.com for the development NOC and support.updatelogic.com for the production NOC. Often called simply "the NOC".
OAD	Over Air Download (update via terrestrial RF broadcast).
OOB Testing	Out Of Box Testing
Open Cable	Standards organization setting standards for Open Cable networks and related technology. http://www.opencable.com
OTA	Over The Air (i.e. Terrestrial Broadcast).
OUI	Organizational Unique Identifier
PSI	Program Specific Information. Generally refers to the PAT, PMT, CAT, NIT, and TDT tables in an MPEG stream.
PSIP	Program and System Information Protocol. Generally refers to the STT, MGT, VCT, RRT, EIT, ETT, and DCCT tables in an MPEG stream.
Publisher	A command line tool provide by UpdateLogic for the purpose of packaging a set of modules for software distribution.
Registration	The process that all devices go through when they first connect to the Internet after shipment in order to register their identity with the UpdateLogic NOC and get a network identity. During this process the device's serial number is sent to the NOC and it receives a ULID that is used for all subsequent conversations with the NOC.
Re-registration	A process that can be initiated via the NOC interface by support staff in order to force a device to re-register and receive its security credentials again.
RTOS	Real Time Operating System.
Security Credential	A device-unique or model group-common file that a device must be provisioned with in order to use the associated streaming service that relies on it.
SecureTV	A facility that delivers streaming media security credentials to a connected device.
Streaming Service	An application that provides streaming video or audio.
StreamViewer	A software tool supplied by UpdateLogic to view and analyze file based transport streams that contain compatible carousels.

SupportTV	A facility that allows remote desktop access to a connected CE device to help trouble shoot customer problems.
SV	Software Version
SVR	Software Version Range
TestCarouselBuilder (TCB)	A Win32 GUI based tool that creates test carousel transport streams from template test files and Publisher packaged software updates.
Update	An entity containing multiple components that may be software or data that replaces the components that were shipped with the TV. Used to fix bugs and add features.
UpdateLogic Agent	Software embedded in a connected device that enables the UpdateTV, SupportTV, and SecureTV facilties.
UpdateTV	A facility that enables device updates via the internet, over-the-air download OAD, or USB/DVD.
UpdateTV OAD Network	A collection of networked data insertion servers, and related broadcast equipment, used to distribute software updates to UpdateTV enabled receivers. The network provides for distribution via over-the-air and Open Cable digital networks environments.
UpdateTV Receiver	A digital television receiver enabled with the UpdateTV Agent so as to receive software updates from the UpdateTV OAD Network.
UpdateTV Server	A computing device used to generate a data stream that can be multiplexed with other digital program content at a broadcast facility for inclusion in the ATSC broadcast stream.
Unit Under Test (UUT)	The CEM's target receiver that is being tested.

INTRODUCTION

This document contains both a "quick start" introduction to porting the Agent as well as a more detailed overview of the source and its function. The quick start section is provided for engineers who, for whatever reason, want to port the Agent as quickly as possible without considering the organization of the source code. The rest of the document takes a more methodical look at how the source is laid out and how it works. A section at the end discusses migration issues from previous versions. Please use any combination of the quick start instructions, reviewing the detailed source code organization, and checking the migrating from previous versions sections that works for you. If you have any questions about this process please contact UpdateLogic customer support for assistance.

QUICK START PORTING INSTRUCTIONS

The following section provides "quick start" instructions for porting the Agent source to your device. The build that you receive comes pre-configured for your OUI and initial model group. Therefore it's possible to make a lot of progress in a very short amount of time.

BUILDING A TEST VERSION OF THE AGENT

The default Makefile assumes you will be building in an x86 Linux 32-bit environment. This is a very good place to become acquainted with the build. You can move onto cross-development in the next section. All of the generic project specific parameters like your OUI, Model Group, and the RSA private key have been prebuilt and included in the tar file your received the Agent source distribution in. Here is the simplest path to exercising the source and becoming acquainted with it.

1. Copy the distribution tar onto an x86 Linux system.
2. Issue a "tar xvf updatetv-agent-2.0.X-Y.tar" command.
3. Issue a "cd updatetv-agent-2.0.X-Y/test" command.
4. Issue a "make" command.
5. Invoke the resulting test program "./test-agent-project"

The test-agent-project test utility doesn't take any parameters. It is driven by key input. Hit the SPACE bar or '?' for a description of what each of the keys does. Test-agent-project incorporates as much project integration related functionality as possible. It is really just a shell program that is simulating events and mocking up a user interface. All of the real functionality that test-agent-project exercises is found in the adaptation file project/project-XXXX.c where XXXX depends on the chipset you're integrating into.

EXERCISING TEST-AGENT-PROJECT

Test-agent-project is a menu driven application which allows access to the different functionality of the Agent. To access the list of available commands use '?' or press ENTER.

NOTE: During test-agent-project execution the output is displayed to the console window in addition to being logged to a file called test.log. There are times during the execution where the Agent encounters warnings and errors that are logged. Some of these warnings and errors are harmless. For example, if the device has not yet registered with the NOC, the Agent attempts to locate certain files such as "./persistent/ulid.dat" which will not be found. Once the device registers with the NOC this file will be available and the warnings and errors associated with the file no longer occur.

The first command to exercise is 'd'. This will display the status of the device. It's a useful debugging function and its UI should be ported to a menu (which can be hidden) on the production version of the device. You will see something like the following when you hit 'd':

```
UTV Thu Nov 19 11:12:27 (main) ESN: SB001
UTV Thu Nov 19 11:12:27 (main) UID: 0
UTV Thu Nov 19 11:12:27 (main) QH:  extdev.updatelogic.com
UTV Thu Nov 19 11:12:27 (main) VER: Base Image
UTV Thu Nov 19 11:12:27 (main) INF: component manifest
```

```
UTV Thu Nov 19 11:12:27 (main) OUI: 0xXXXXXX
UTV Thu Nov 19 11:12:27 (main) MG:  0xXXXX
UTV Thu Nov 19 11:12:27 (main) HM:  0x0001
UTV Thu Nov 19 11:12:27 (main) SV:  0x0001
UTV Thu Nov 19 11:12:27 (main) MV:  0x0000
UTV Thu Nov 19 11:12:27 (main) NP:  0
UTV Thu Nov 19 11:12:27 (main) REG: 0
UTV Thu Nov 19 11:12:27 (main) FCT: 0
UTV Thu Nov 19 11:12:27 (main) FET: 0
UTV Thu Nov 19 11:12:27 (main) EC:  0
UTV Thu Nov 19 11:12:27 (main) LE:  0
UTV Thu Nov 19 11:12:27 (main) LES: normal (success) return
UTV Thu Nov 19 11:12:27 (main) LET: Wed Dec 31 19:00:00 1969
```

The next functionality to test is the response to a "network up" event. When the network on a device first comes up a connection to the ULI NOC is made. If it's the first connection the device has made to the ULI NOC then registration and initial provisioning takes place at this time. If it's a connection made after initial registration then the SecureTV local cache is updated at this time. Use the 'n' command to simulate a network up event which will kick off registration and provisioning.

QUICK TEST OF THE UPDATE MECHANISM

The next functionality to test is an internet download. Use the 'p' command to simulate a power down event which should kick off registration, provisioning, and the download of an update.

You will then have to simulate rebooting the Agent to display the new status. You can do this by quitting ('q') and restarting the test-agent-project program or by typing 's' for shutdown and then 'b' for boot. If you then type 'd' you will see that the device has been updated. The items in red have changed.

```
UTV Thu Nov 19 11:12:27 (main) ESN: SB001
UTV Thu Nov 19 11:12:27 (main) UID: 0
UTV Thu Nov 19 11:12:27 (main) QH:  extdev.updatelogic.com
UTV Thu Nov 19 11:12:27 (main) VER: XXXX (depends on your build)
UTV Thu Nov 19 11:12:27 (main) INF: Test Update
UTV Thu Nov 19 11:12:27 (main) OUI: 0xXXXXXX
UTV Thu Nov 19 11:12:27 (main) MG:  0xXXXX
UTV Thu Nov 19 11:12:27 (main) HM:  0x0001
UTV Thu Nov 19 11:12:27 (main) SV:  0x0002
UTV Thu Nov 19 11:12:27 (main) MV:  0x0002
UTV Thu Nov 19 11:12:27 (main) NP:  1
UTV Thu Nov 19 11:12:27 (main) REG: 1
UTV Thu Nov 19 11:12:27 (main) FCT: 0
UTV Thu Nov 19 11:12:27 (main) FET: 0
UTV Thu Nov 19 11:12:27 (main) EC:  0
UTV Thu Nov 19 11:12:27 (main) LE:  0
UTV Thu Nov 19 11:12:27 (main) LES: normal (success) return
UTV Thu Nov 19 11:12:27 (main) LET: Wed Dec 31 19:00:00 1969
```

If you run the test again you will see that that the results of the UpdateQuery command are that there aren't any updates available. That's because the device has already taken he only update that's available to it. In order to force the device to re-take the initial update two things must be done. The first is that the device itself must have its identity reverted. The second is that the NOC must be told that the device should be re-registered.

To revert the identity of the device hit 'r' for reset which will copy the "backup" component manifest over the "active" component manifest. If you do this the device still will not take updates because the NOC doesn't know about the local change. In order to force the NOC to take notice of the local identity change log into the NOC using the provided credentials and navigate to Internet Download->Change Device Status. Enter the serial number of the device into serial # field. You can find the serial number of the device in the file persistent/snum.txt. Hit the Show Device button and then click the re-register radio button and hit the Update Status button. You will be prompted to make sure you really want to do this and the Update Status button will turn into an Update Now button. Hit the Update Now button and the NOC will then force the device to re-register when it next makes contact. Type the 'p' key again and the device should download the update again.

Please note that you always have to simulate power down and power up in order to install the new update's identity.

Now let's try the same thing using a simulated USB update. A USB update contains the same data that's delivered via Internet, but it's contained in a static file that's accessible locally. This file is called a UTV file.

Revert the identity of the device using the 'r' command. Now type 'v'. (This exercises the DVD interface because it happens to be more simple than a USB device, but it functions identically). When you type 'v' the update should take place. In this case the NOC wasn't involved because the update was served locally directly from a file.

There are many other commands that the project simulation is capable of. They're described in the *Project Simulation Commands* section below.

BUILDING THE AGENT FOR YOUR TARGET

The source release you received contains a private library built for the SoC running on your target platform. When you executed the steps above you used the built-in x86 Linux version of the private library. For your actual target platform you have two choices.

1. Examine the Agent Makefile to see what files it uses and import those files and those build steps into your build environment or…

2. Use the Makefile as is to quickly get a test program working.

If you would like to try option 2 then you will need to customize the Makefile to your target environment. To use the private library for your target edit the Makefile and change the PLATFORM_BUILD_ENV from

linux_x86 to XXX where XXX is the target build environment described in the Readme.html document you received with the build. You will then need to wire your tool chain into the make. Find the line that says:

```
# Set your cross-dev compiler HERE:

ifeq "$(PLATFORM_BUILD_ENV)" "XXXXXX"

CXX = $(PREFIX)mipsel-linux-gcc

endif
```

Replace “mipsel-linux-gcc” with your target’s compiler.

Then issue a “make” and re-run the tests using test-agent-project above. If you can get those to work then you are ready to perform the real integration steps below.

LOCATION AND FUNCTION OF CONFIGURATION FILES

This section describes the name, function, and define that controls the location of each of the paths and configuration files in the Agent. These defines all live inside of project-XXXX.h.

Path Configuration

Define	Default value	Description
UTV_PLATFORM_PERSISTENT_PATH	./persistent	This directory that contains all files that the Agent writes. Must be in a r/w partition.
UTV_PLATFORM_INTERNET_PROVISIONER_PATH	./persistent/provisioner	A sub-dir of the persistent directory that contains the local SecureTV cache.
UTV_PLATFORM_INTERNET_FACTORY_PATH	./factory	This directory that contains all files that must be inserted at the factory. Should be mounted r/w in the factory, but r/o at all other times
UTV_PLATFORM_INTERNET_READ_ONLY_PATH	./read-only	This directory contains r/o files that are typically provided in a cramfs.img.

File Name Configuration(the items marked in red only apply to UpdateTV ports)

Define	Path	File Name	Description
UTV_PLATFORM_INTERNET_ULID_FNAME	Persistent	ulid.dat	Network identity file written when the device registers with the NOC.
UTV_PLATFORM_INTERNET_ULPK_FNAME	Factory	ulpk.dat	Device-unique key data file inserted at the factory.
UTV_PLATFORM_SERIAL_NUMBER_FNAME	Factory	snum.txt	A place holder file that contains serial numbers. Expected to be replaced by device-specific serial number handling.
UTV_PLATFORM_INTERNET_SSL_CA_CERT_FNAME	Read Only	uli.crt	X.509 CA root cert for UTV network conversations.
UTV_PLATFORM_UPDATE_STORE_0_FNAME	Persistent	update-0.store	Update candidate storage area 0.
UTV_PLATFORM_UPDATE_STORE_1_FNAME	Persistent	update-1.store	Update candidate storage area 1.
UTV_PLATFORM_ACTIVE_COMP_MANIFEST_FNAME	Persistent	component.manifest	Active component manifest that device takes its compatibility identity from.
UTV_PLATFORM_BACKUP_COMP_MANIFEST_FNAME	Read Only	component.manifest	Backup component manifest that seeds initial active c.m. on a virgin system. Copied to active location on first boot.
UTV_PLATFORM_FACTORY_COMP_MANIFEST_FNAME	Read Only	component.manifest.factory	Factory component manifest that is used during out of box testing. Puts device into special test mode.
UTV_PLATFORM_PERSISTENT_STORAGE_FNAME	Persistent	persistent-storage.dat	Fixed length file that saves device state and stores error and performance events.
UTV_PLATFORM_KEYSTORE_FNAME	Read Only	keystore.dat	RSA private key for the device class.
UTV_TEST_LOG_FNAME	Test	test.log	Output log file
UTV_PLATFORM_TEST_TS_FNAME	Test	Stream.ts	Test TS file input.
UTV_PLATFORM_UPDATE_REDIRECT_FNAME	N/A	updatelogic.txt	Media-delivered (USB/DVD) update re-direct file. Contains name of UTV file that contains update.
UTV_PLATFORM_REGSTORE_STORE_AND_FWD_FNAME	Persistent	regstore.txt	Registration store and forward file.

ADAPTATION FUNCTIONS

This section describes the platform adaptation functions that are contained in the file project-XXXX.c. These functions *must* be adapted by the integrator to perform platform-specific versions of the functions they represent. The operational goal and placeholder function of each of these entry points is described. Please note that function names in red are not needed for a SecureTV only port.

GENERAL ADAPTATION FUNCTIONS

UTV_RESULT UtvCEMGetSerialNumber(UTV_INT8 *pubBuffer, UTV_UINT32 uiBufferSize)

- This function is used to retrieve the platform's serial number.
- The placeholder code opens the file pointed at by `UTV_PLATFORM_SERIAL_NUMBER_FNAME` and returns its contents.
- Replace the body of this function with code to retrieve the serial number from wherever it's stored.

static UTV_UINT16 UtvCEMReadHWMFromRegister(void)

- This function returns the hardware model of the device. The hardware model is an index from 1 to n of all of the different hardware configurations in a given model group *that share the same update image.* Hardware model differentiation is typically applied to panel sizes. The 42" panel will be hardware model 1. The 48" panel will be hardware model 2, and so on.
- The placeholder code just returns the value 1.
- Replace the body of this function with code to read the panel size from a hw register or an index that is stored in NVRAM during factory setup.

UTV_UINT16 UtvCEMTranslateHardwareModel(void)

- This function translates the value returned from UtvCEMReadHWMFromRegister() into a NOC-registered hardware model value. The defines UTV_CEM_HW_MODEL_X that are returned are contained in file cem-XXXXXX-YYYY.h where XXXXXX is the OUI of the CEM and YYYY is the model group.
- The placeholder code assumes you have 3 hardware model groups.
- Replace the body of this function with a case statement that reflects the number of hardware models in the model group for the device you're creating.

UTV_RESULT UtvPlatformGetEtherMAC(UTV_INT8 *pszMAC, UTV_UINT32 uiBufferSize)

- This function returns the Ethernet MAC address as a string in the provided buffer.

- The placeholder code copies a canned MAC address from the define S_FAKE_ETHERNET_MAC contained in the file project-XXXX.c
- Replace the body of this function with code to return the actual Ethernet MAC address of the device.

UTV_RESULT UtvPlatformGetWiFiMAC(UTV_INT8 *pszMAC, UTV_UINT32 uiBufferSize)

- This function returns the Wifi MAC address as a string in the provided buffer.
- The placeholder code copies a canned MAC address from the define S_FAKE_WIFI_MAC contained in the file project-XXXX.c
- Replace the body of this function with code to return the actual WiFi MAC address of the device.
- *EXCEPTION*: Only needed on devices with Wifi. Leave code in place to return fake WiFi MAC if your device does not support it.

UTV_BOOL UtvPlatformSystemTimeSet(void)

- This function returns a bool indicating whether the system time has been set via NTP, STT, or the consumer.
- The placeholder code return false.
- Replace the body of this function with code to sense whether system time has been set via any of the methods described. If the device has a real-time clock then always return TRUE unless there has been a power outage or other event that could create an invalid time.

UTV_RESULT UtvPlatformFactoryModeNotice(UTV_BOOL bFactoryMode)

- This function is called by the Agent with the factory mode state.
- The placeholder code displays a simple string if factory mode is true.
- Replace the body of this function with code that displays some sort of warning like text in an overlay in the graphics plane so that devices are never shipped in factory mode.

UTV_RESULT UtvPlatformCommandInterface(UTV_INT8 *pszCommand)

- This function is used by the Agent to allow secure commands to be executed by the device. The command set is entirely up to the CEM. These commands are created by special entries in Publisher script files. Please see the *UpdateLogic Publisher* document for more information.
- The placeholder function just displays the command that would be executed.
- Replace the body of this function with code that executes the command provided. This can be as simple as spawning the command through the Linux "system" command or as complex as you would like to make it. Typically access to the Linux "system" command and diagnostic and development functions are supported.

GENERAL UPDATE INSTALLATION ADAPTATION FUNCTIONS

`void * UtvPlatformInstallAllocateModuleBuffer( UTV_UINT32 iModuleSize )`

- This function is used by the Agent to allocate a large buffer to decrypt a received update module into. The size of this allocation is typically 2MB.
- The placeholder code calls UtvMalloc() which in turn just calls malloc().
- Replace the body of this function with code to allocate a 2MB chunk of RAM. Many modern devices simply use the placeholder function as is. If your device is particularly RAM-constrained you can do things like allocate video buffer memory or other special purpose RAM. Please make sure to turn off all active DMAs to special purpose RAM so that they don't interfere with the use of the allocated buffer!

`UTV_RESULT UtvPlatformInstallFreeModuleBuffer( UTV_UINT32 iModuleSize, void * pModuleBuffer )`

- This function is the de-allocation companion to the function above. It returns memory allocated by UtvPlatformInstallAllocateModuleBuffer().
- The placeholder code calls UtvFree() which in turn calls free().
- Replace the body of this function only if you resorted to special allocation handling in UtvPlatformInstallAllocateModuleBuffer().

`UTV_RESULT UtvPlatformInstallComponentComplete( UTV_UINT32 hImage, UtvCompDirCompDesc *pCompDesc )`

- This function is called when all the modules for a given component regardless of its partition/file attribute have been received.
- The placeholder code displays a text message containing the component name.

- Replace the body of this function with code that handles the component complete case. This may include proprietary validation or installation functionality.

UTV_RESULT UtvPlatformInstallUpdateComplete(UTV_UINT32 hImage)

- This function is called when all components in the update have been installed.
- The placeholder code contains calls to functions *which must be included.* These calls include UtvImageValidateHashes() and UtvManifestCommit().
- Add code to the *end* this function to perform platform-specific update functionality. This should include changes to platform-specific persistent memory that is checked on boot to see if an update has taken place. Typically the code added to this function does not initiate the actual switch from the currently active code/data to the newly installed code/data. Instead it informs the platform that an update is available and on the next boot platform-specific code switches to the available update.

UTV_RESULT UtvPlatformInstallCopyComponent(UTV_UINT32 hImage, UtvCompDirCompDesc *pCompDesc)

- This function is called when a component in an update does not need to be updated due to having an incompatible version. This is typically the case when the installed component is already at the version of the potential update component. This function is used to copy the current component which will not be updated to the "next" update set partition.
- The placeholder code just displays a message indicating the function has been called.
- Replace the body of this function with code to copy the named component from its "active" location to its "next" location.

PARTITION INSTALLATION ADAPTATION FUNCTIONS

UTV_RESULT UtvPlatformInstallPartitionOpenWrite(UTV_UINT32 hImage, UtvCompDirCompDesc *pCompDesc, UtvModSigHdr *pModSigHdr, UTV_UINT32 *puiHandle)

- This function is called during an update when the Agent first encounters a module from a component whose partition attribute is true. The hImage argument describes which internal "store" the component directory for the component is located in. The pCompDesc argument points to a UtvCompDirCompDesc structure (defined in utv-image.h) that contains a description of the component, including most importantly its name which may be used as the partitions name. The pModSigHdr argument points to a UtvModSigHdr structure (defined in utv-image.h) that contains information about the module itself including its size, etc. The puiHandle points to a instance handle returned by this function

which is passed to the functions UtvPlatformInstallPartitionWrite() and UtvPlatformInstallPartitionClose().

- The placeholder code provides an example for how to treat the component name as a partition name and simulates opening that partition.
- Replace the body of this function with code that actually calls the correct MTD management functions on your platform to open the specified partition for write.

UTV_RESULT UtvPlatformInstallPartitionOpenRead(UTV_UINT32 hImage, UtvCompDirCompDesc *pCompDesc, UTV_BOOL bActive, UTV_UINT32 *puiHandle)

- This function is called during validation of a partition that was previously written with the functions UtvPlatformInstallPartitionOpenWrite() , UtvPlatformInstallPartitionWrite() and UtvPlatformInstallPartitionClose(). The hImage argument describes which internal "store" the component directory for the component is located in. The pCompDesc argument points to a UtvCompDirCompDesc structure (defined in utv-image.h) that contains a description of the component, including most importantly its name which may be used as the partition's name. The bActive argument is reserved. The puiHandle points to a instance handle returned by this function which is passed to the functions UtvPlatformInstallPartitionRead() and UtvPlatformInstallPartitionClose().
- The placeholder code performs an open on the partition simulation file.
- Replace the body of this function with code that calls the correct MTD management functions on your platform to open the specified partition for read.

UTV_RESULT UtvPlatformInstallPartitionWrite(UTV_UINT32 uiHandle, UTV_BYTE *pData, UTV_UINT32 uiDataLen, UTV_UINT32 uiDataOffset)

- This function is called during an update after UtvPlatformInstallPartitionOpenWrite() to write update modules to the currently open partition. The uiHandle argument will be the same handle returned by the UtvPlatformInstallPartitionOpenWrite() call. The pData argument points to the data to write. The uiDataLen argument is the length of that data. The uiDataOffset is the offset into the partition that the data should be written.
- The placeholder code performs a write on the partition simulation file.
- Replace the body of the function with code that calls the correct MTD write function on your platform to write the data specified at the offset specified.

UTV_RESULT UtvPlatformInstallPartitionRead(UTV_UINT32 uiHandle, UTV_BYTE *pBuff, UTV_UINT32 uiSize, UTV_UINT32 uiOffset)

- This function is called during validation of a partition that was previously opened with the function UtvPlatformInstallPartitionOpenRead(). The uiHandle argument was returned by UtvPlatformInstallPartitionOpenRead(). The pBuff argument points to a buffer that is

uiSize bytes in length. The uiOffset argument specifies the offset into the partition to read the uiSize bytes from.

- The placeholder code performs a read on the partition simulation file.
- Replace the body of the function with code that calls the correct MTD function on your platform to read the data specified at the offset specified.

```
UTV_UINT32 UtvPlatformInstallPartitionClose( UTV_UINT32 uiHandle )
```

- Closes partitions previously opened for write or read by UtvPlatformInstallPartitionOpenWrite() or UtvPlatformInstallPartitionOpenRead(). The uiHandle argument should have been returned by one of those functions.
- The placeholder code closes the partition simulation file.
- Replace the body of the function with code that calls the correct MTD function on your platform to close the partition specified by the handle.

FILE INSTALLATION ADAPTATION FUNCTIONS

```
UTV_RESULT UtvPlatformInstallFileOpenWrite( UTV_UINT32 hImage,
UtvCompDirCompDesc *pCompDesc, UtvModSigHdr *pModSigHdr, UTV_UINT32
*puiHandle )
```

- This function is called during an update when the Agent first encounters a module from a component whose partition attribute is *false*. The hImage argument describes which internal "store" the component directory for the component is located in. The pCompDesc argument points to a UtvCompDirCompDesc structure (defined in utv-image.h) that contains a description of the component, including text fields that can be customized to point to file names, etc. The pModSigHdr argument points to a UtvModSigHdr structure (defined in utv-image.h) that contains information about the module itself including its size, etc. The puiHandle points to a instance handle returned by this function which is passed to the functions UtvPlatformInstallFileWrite() and UtvPlatformInstallFileClose().
- The placeholder code provides an example for how to treat the component name as a file and opens that file for write.
- Replace the body of this function with code that actually calls the correct file management functions on your platform to open the specified file for write. The UtvPlatformFileOpen() call is perfectly suitable for this.

```
UTV_RESULT UtvPlatformInstallFileOpenRead( UTV_UINT32 hImage,
UtvCompDirCompDesc *pCompDesc, UTV_BOOL bActive, UTV_UINT32 *puiHandle  )
```

- This function is called during validation of a file that was previously written with the functions UtvPlatformInstallFileOpenWrite() , UtvPlatformInstallFileWrite() and UtvPlatformInstallFileClose(). The hImage argument describes which internal “store” the component directory for the component is located in. The pCompDesc argument points to a UtvCompDirCompDesc structure (defined in utv-image.h) that contains a description of the component, including most importantly its name which may be used as the partition’s name. The bActive argument is reserved. The puiHandle points to a instance handle returned by this function which is passed to the functions UtvPlatformInstallFileRead() and UtvPlatformInstallFileClose().
- The placeholder code performs an open on the simulation file.
- Replace the body of this function with code that calls the correct file management functions on your platform to open the specified file for read. The UtvPlatformFileRead() call is perfectly suitable for this.

```
UTV_RESULT UtvPlatformInstalFileWrite( UTV_UINT32 uiHandle, UTV_BYTE
*pData, UTV_UINT32 uiDataLen, UTV_UINT32 uiDataOffset )
```

- This function is called during an update after UtvPlatformInstallFileOpenWrite() to write update modules to the currently open file. The uiHandle argument will be the same handle returned by the UtvPlatformInstallFileOpenWrite() call. The pData argument points to the data to write. The uiDataLen argument is the length of that data. The uiDataOffset is the offset into the file that the data should be written. Please note that if your platform is using Broadcast download then modules can come out of order. With Internet and File (USB) download modules always come in order.
- The placeholder code performs a write on the simulation file.
- Replace the body of the function with code that calls the correct function on your platform to write the data specified at the offset specified.

```
UTV_RESULT UtvPlatformInstallFileRead( UTV_UINT32 uiHandle, UTV_BYTE
*pBuff, UTV_UINT32 uiSize, UTV_UINT32 uiOffset )
```

- This function is called during validation of a partition that was previously opened with the function UtvPlatformInstallFileOpenRead(). The uiHandle argument was returned by UtvPlatformInstallFileOpenRead(). The pBuff argument points to a buffer that is uiSize bytes in length. The uiOffset argument specifies the offset into the file to read the uiSize bytes from.
- The placeholder code performs a read on the simulation file.
- Replace the body of the function with code that calls the correct function on your platform to read the data specified at the offset specified.

UTV_UINT32 UtvPlatformInstallFileClose(UTV_UINT32 uiHandle)

- Closes files previously opened for write or read by UtvPlatformInstallFileOpenWrite() or UtvPlatformInstallFileOpenRead(). The uiHandle argument should have been returned by one of those functions.
- The placeholder code closes the simulation file.
- Replace the body of the function with code that calls the correct function on your platform to close the file specified by the handle. UtvPlatformFileClose() is perfectly suited for this.

SECURITY ADAPTATION FUNCTIONS

UTV_RESULT UtvPlatformSecureOpen(void)

- Is called once on power up via UtvPublicAgentInit().
- The placeholder code does nothing but return UTV_OK.
- Replace the body of the function with code that acquires and store instance handles for the other security-related functions that follow..

UTV_RESULT UtvPlatformSecureClose(void)

- Is called once on power down via UtvPublicAgentShutdown().
- The placeholder code does nothing but return UTV_OK.
- Replace the body of the function with code that releases any resources allocated by UtvPlatformSecureOpen().

UTV_RESULT UtvPlatformSecureRead(UTV_INT8 *pszFName, UTV_BYTE *pBuff, UTV_UINT32 uiBufferSize, UTV_UINT32 *puiDataLen)

- Is designed to be called to read a specified encrypted file from flash memory and return the decrypted data in a buffer. Is the companion function to UtvPlatformSecureWrite(). Is called by various parts of the UpdateLogic Agent to read back data that was protected by the UtvPlatformSecureWrite() call. The pszFName argument points to a zero terminated string containing the path and file name of the file to read. The pBuff argument points to the buffer to read the data into. The uiBufferSize argument is the length of the buffer to receive the data. The puiDataLen argument is a pointed to the actual length of the data decrypted from the file.
- Placeholder function assumes that the file being read does not have a valid encryption header and is therefore not encrypted. The placeholder function obviously cannot perform any hardware-based decryption.
- This function should employ the SoC's hardware protected key to securely decrypt the file provided. The file that this function is targeted to decrypt should not be decrypt-able except

on the device that it was encrypted on. If the platform supports a secure file partition that decrypts on the fly this file may simply be read back from that partition.

UTV_RESULT UtvPlatformSecureWrite(UTV_INT8 *pszFileName, UTV_BYTE *pubData, UTV_UINT32 uiSize)

- Is designed to take a buffer of in-the-clear data, encrypt it, and write it out as the specified file. Is the companion function of UtvPlatformSecureRead(). Is called by various parts of the UpdateLogic Agent to protect sensitive data that must persist between power cycles. This data will later be read by UtvPlatformSecureRead(). The pszFName argument points to a zero terminated string containing the path and file name of the file to written. The pubData argument points to the buffer to encrypt and write. The uiSize argument is the length of the data to be encrypted and written.
- Placeholder function writes the file in an unencrypted fashion.
- This function should employ the SoC's hardware protected key to securely encrypt the data provided and write it flash memory. The file that this function is targeted to encrypt should not be decrypt-able except on this same device. If the platform provides a secure file partition that encrypts on the fly then this file may simply be written into that partition.

UTV_RESULT UtvPlatformSecureDeliverKeys(void *pKeys)

- Called by the update payload decryption code to deliver SoC-specific keys that are embedded in the encrypted payload header of an update module. These keys should be stored in memory (not flash) for later use by the UtvPlatformSecureDecrypt() function.
- Placeholder function does nothing. The pKeys pointers points to an SoC-specific key(s).
- Replace the body of this function with code to store the delivered key(s) so that it can be used by UtvPlatformSecureDecrypt(). The number, size, and structure of these keys is platform dependent and should be coordinated with ULI so that the Publisher utility can deliver the appropriate key format. If your platform doesn't have platform-specific keys that are delivered with a payload then this function can be a NOP.

UTV_RESULT UtvPlatformSecureDecrypt(UTV_UINT32 uiOUI, UTV_UINT32 uiModelGroup, UTV_BYTE *pInBuff, UTV_BYTE *pOutBuff, UTV_UINT32 uiDataLen)

- Called by the ULI private library when the update payload decryption code detects that it is supposed to use hardware accelerated decryption of a chunk of payload data. Typically these chunks are about 2MB long. This function will only be called when Publisher uses the –x option to "externally" encrypt the update payloads with a format that the hardware is capable of decrypting. The uiOUI argument contains the OUI of the update that is being decrypted. The uiModelGroup argument contains the model group of the update that is being decrypted. The pInBuff argument points to the data to be decrypted. The pOutBuff argument points to the buffer to receive the data that has been decrypted. The uiDataLen contains the length of the data to be decrypted. It is assumed that the length of the encrypted data is the same as the length of the decrypted data.

- Placeholder function does nothing.
- This function should use the key(s) provided by UtvPlatformSecureDeliverKeys() to perform hardware-accelerated decryption of the buffer pointed to by pInBuff and return it in memory pointed to by pOutBuff. If your platform doesn't perform hardware-assisted payload decryption then this function can be a NOP.

```
UTV_RESULT UtvPlatformSecureGetULPK(UTV_BYTE **pULPK, UTV_UINT32
*puiULPKLength )
```

- Called by the ULI private library to return a decrypted version of the ULPK. ULPK encryption is platform-specific. ULI will work with the integrator to create an encrypt/decrypt model that takes advantage of the platform's crypto block features.
- Placeholder function assumes that the ULPK is a file that can be decrypted using UtvPlatformSecureRead(). This will only work on platforms that do not use a secure file partition for on the fly encrypt/decrypt.
- This function should use platform-specific code to decrypt the ULPK. The ULPK is typically encrypted with a hardware protected pre-shared key so this function should send the ULPK to the crypto block, decrypted it, and return the decrypted contents of the ULPK.

BROADCAST DELIVERY ADAPTATION FUNCTIONS

Adaptation of these function is only required if your device uses broadcast update delivery. If it doesn't then no changes need to be made to these functions.

```
UTV_RESULT UtvGetTunerFrequencyList ( UTV_UINT32 * piFrequencyListSize,
UTV_UINT32 * * ppFrequencyList )

UTV_RESULT UtvPlatformDataBroadcastAcquireHardware(UTV_UINT32
iEventType)

UTV_RESULT UtvPlatformDataBroadcastReleaseHardware()

UTV_RESULT UtvPlatformDataBroadcastSetTunerState( UTV_UINT32 iFrequency )

UTV_UINT32 UtvPlatformDataBroadcastGetTunerFrequency( void )

UTV_RESULT UtvPlatformDataBroadcastGetTuneInfo( UTV_UINT32
*puiPhysicalFrequency, UTV_UINT32 *puiModulationType )

UTV_RESULT UtvPlatformDataBroadcastCreateSectionFilter(
UTV_SINGLE_THREAD_PUMP_FUNC *pPumpFunc )

UTV_RESULT UtvPlatformDataBroadcastOpenSectionFilter( UTV_UINT32 iPid,
UTV_UINT32 iTableId, UTV_UINT32 iTableIdExt, UTV_BOOL bEnableTableIdExt )

UTV_RESULT UtvPlatformDataBroadcastCloseSectionFilter( )

UTV_RESULT UtvPlatformDataBroadcastDestroySectionFilter( )
```

OTHER ADAPTATION FUNCTIONS

There are many other functions in the project-xxxx.c file. These functions should not have to be adapted to be used on your platform. They're provided to improve understanding and create an at-a-glance view of the project-related functions that are operating in your integration.

OTHER ADAPTION REQUIREMENTS

SIGNALS

The application using the Device Agent should catch and handle (or scuttle) expected signals. Because the Device Agent uses network communication, and most often SSL network communication, the SIGPIPE signal can be raised. For example, by an underlying secure network communication library such as OpenSSL. If the application does not catch SIGPIPE it may be terminated by the run-time environment for expected occurrences of SIGPIPE.

FUNCTIONAL OVERVIEW

NetReady 3.0 provides a comprehensive update and provisioning solution for connected appliances. Its design and feature set is informed by UpdateLogic's (ULI) many years of experience integrating secure update technology into consumer appliances. NetReady 3.0 is designed to assist original electronics manufacturers (OEM) in solving real-world product development, manufacturing, and support issues.

NetReady 3.0 functionality is divided into three main facilities: UpdateTV, SecureTV, and SupportTV.

UPDATETV

UpdateTV is the name of the software update facility. UpdateTV breaks your appliance's update needs down into *components.* A component is an independently addressable update entity. Examples of components are root file systems, kernels, splash screens, panel-specific picture quality tuning parameters, boot ROMs, applications, data files, retail demo loops, etc. Components may either be partitions or files. There is no practical limit on the size or number of components in a given update.

In order to simplify configuration options and improve quality assurance and support all compatible devices are kept at a same update level. Although each update contains all the latest components, from update to update only the components that have changed are updated on a given device.

Each update may be delivered by any combination of UpdateLogic's nationwide over-the-air broadcast network, the internet, or USB stick. Updates are built on industry-standard encryption and authentication technology so your device's software cannot be tampered with no matter how it is delivered.

SECURETV

SecureTV is the name of the security credential delivery facility. Connected appliances require certificates, encryption keys, and other objects that are used in digital rights management of streaming media services. SecureTV securely delivers these valuable credentials to an appliance in a manner that deters theft by keeping them off of the factory floor, ties them to specific known-secure versions of software, and allows consumer electronics manufacturers to work in cooperation with streaming media

providers to manage service access. If you're performing a SecureTV only port then please skip to the *SecureTV Only Ports* section at the end of this document.

SUPPORTTV

SupportTV is the name of the live remote support facility. It is a product offering that leverages UpdateLogic's field device communications infrastructure to reduce product support costs for device vendors. SupportTV features the mechanisms to provide live Internet remote assistance sessions to field devices for support technicians. The remote sessions receive audio/video output and send device control input.

THE AGENT, THE NOC, AND PUBLISHER

The UpdateTV software that runs on an appliance is called the *Agent.* The UpdateLogic servers that control the dissemination of updates to appliances are called the *NOC* (Network Operating Center). The *Publisher* tool takes the raw output of an appliance's software build process and encrypts them and wraps them in a compatibility envelope. The output of Publisher is uploaded to the NOC.

The Agent is responsible for checking for updates via the 3 delivery methods, communicating with the NOC to receive internet updates and SecureTV credentials, convey status information, and interact with the appliances supervising program to install updates and inform the consumer about update status, etc. The NOC is the repository for all updates and provisioned credentials, keeps track of versioning, manages the nationwide broadcast update network, uploads internet updates to Content Delivery Networks (CDNs), and is the hub through which all updates are controlled. For more information on the NOC please consult the *UpdateTV Network Operations Center CEM Users Guide.pdf*. For more information about Publisher please consult the *UpdateLogic Publisher.pdf* document.

A useful overview of the product from a technical support perspective can also be found in the document called *NetReady Technical Support Guide.pdf*

A useful overview of the product from a factory perspective can also be found in the document called *UpdateLogic Factory Process Guide.pdf*

UTV AGENT

This section introduces basic concepts of the Agent's design and the implications of this design on different use cases.

OPERATING MODES

The Agent has four basic operating modes:

- Scanning – looking to see if an update is available.
- Downloading – acquiring and storing an update that has been found.
- Querying – presenting stored updates to the user for acceptance in the case of optional components or acknowledgement in the case of required components.

- Installing – installing an update that has been stored and approved or acknowledged by the user.

Each of these modes affects the system and user in different ways.

DELIVERY MODES

In addition to the four basic operating modes there are three delivery modes. The operation of Scanning and Downloading differ for each of these delivery modes due to their physical restrictions, but the Agent interfaces to operate them are the same. Installation is not affected by delivery mode. The delivery modes are:

- Internet – the Agent makes contact with the NOC to acquire updates over an internet connection. The nature of this connection, wireless or cabled, doesn't matter. This delivery mode can be used while the system is in use as long as arbitration is performed to avoid sluggish internet response when the user is performing internet-related tasks.
- Broadcast – the Agent tunes an OTA or Cable frequency containing an update carousel. This delivery mode requires access to a tuner. In a two tuner system this can be done while the TV is in use if the second tuner is available. Typically this delivery mode is used when the TV panel is off to avoid tuner conflict.
- File – the Agent is given a file to use as an update via either USB or download. The user is actively involved in manually providing the update so the operation of the TV is directly impacted.

UPDATE TYPES

There are two basic update types: optional and required. Optional updates may include feature extensions or additions and minor bug fixes. Because the installation of any update may require a significant interruption in the use of the TV these optional updates can be deferred by the user for a given amount of time from 1-X days and may include dismissing it altogether. Required updates do not ask the user for permission. They cannot be deferred. The user is notified that the system needs to be updated and the Agent proceeds to install the update. Updates are packaged in a delivery "envelope" that contains meta-data related to the update including its optional vs. required status and text that can be displayed explaining the update. For more information about this meta-data please see the *UpdateTV Publisher* documentation.

TIMING CONSIDERATIONS

Each operating and delivery mode has different restrictions on when it can be run.

- Immediately after the TV is turned off – Due to EnergyStar requirements the best strategy for conducting Agent operations that would disturb the user is immediately after the TV has been turned off. These typically include broadcast scan, but can also include internet scan or download as long as the scan or download delay is adhered to. After Agent operations are complete the TV can proceed to power-down into standby mode.
- Immediately after the TV has been turned on – Updates that have been stored and not have been previously viewed and deferred by the user (assuming they are optional) should be presented to the user for confirmation as soon as the TV is turned on. If the user decides to accept optional updates or is simply acknowledging a required update the Agent then goes on to

install those updates. This installation is CPU-intensive due to decryption and other TV functions will be limited. The installation may require one or more system reboots.

- Automated wake-up via external timer – Broadcast downloads occur on a regular repeating basis via carousel. Their availability cannot be controlled by the TV. Therefore the TV must wake up at a given time to receive them. This requires an external timer to take the system out of standby at a given time.
- At the discretion of the user – the system should respond to USB inserts or commands to download an update any time the user directs it to.
- Anytime – Some operations like internet scan can be run in the background while the TV is in use (even if the internet connection is active) without disturbing the user or the system. These operations may be done at anytime.

DIVISION OF LABOR BETWEEN THE AGENT AND THE SYSTEM

The Agent runs in what is called *interactive mode*. This means it will *not* wake up on its own and perform any operations unless the supervising TV application ("the System") tells it to. The Agent presents the results of each of its operations back to the System so that the System can decide what to do next. *All interpretation of these results and the user interactions that may follow are done by the System.* The Agent does not have any awareness of menus or user interaction. Its job is to deliver the meta-data associated with each update ahead of a download, allow the System to determine what to download, store the approved downloads, make the meta-data associated with stored downloads available to the System, and when instructed install the stored updates. The System handles the decision making and involves the user as it sees fit.

FIELD VS. FACTORY MODE

Field updates are updates that occur once a TV has been shipped. Factory updates occur in the factory before the TV has been shipped. The Agent is agnostic about the difference between these two modes. Te difference is largely in the special user interface that factory updates use. The Agent must be put in factory mode and the arguments to its public entry points change a little to indicate to the Agent that it is being asked to operate in factory mode, but the overall flow of operation of scanning, downloading, storing, and installing is the same in field and factory modes. The Agent is aware of factory mode because factory updates use a special "light" encryption which decrypts very quickly to save time on the assembly line. The Agent only allows this light encryption when in factory mode.

PACKAGES, UPDATES, COMPONENTS, AND MODULES

A "package" consists of multiple "updates". An "update" consists of multiple "components". "Components" are split into transport envelopes called "modules". By default modules are 2MB in length. Modules are individually encrypted. Updates stored in the TV are stored with a "component directory" followed by multiple components each of which is made up of one or more modules. This storage form is identical to the file format used in USB update distribution. A USB update distribution looks to the system as if an update were stored on a remote device and it treats it the same way it treats all stored updates. Updates are stored in encrypted form. Decryption takes place during installation. The Agent handles the management of all of these containers in a way that hides as much of the

implementation details as possible. Modules, for instance, are not exposed to the user, and are only nominally exposed to the System. Typically, for now, we will only be working with updates, not packages of updates. Components, however, are the basic currency of update organization. Publisher binds a group of components into an update. The Agent can independently access the components within an update. Attributes and arbitrary text fields may be associated on a scope that is global to an update and on a per component basis. This allows the System to treat components differently assigning some components to be optional, for instance, or to have different descriptive text or release notes. The system architect may choose to only have one component in an update called the "system" (for example) and manage the components within that system by itself without relying on the UTV component architecture.

APPLIANCE ADDRESSABILITY AND THE NOC

Every appliance on the UpdateTV internet network is uniquely addressable. An appliance's *address* consists of an *OUI* (CEM's organizationally unique identifier provided by the IEEE), a *model group* (16-bit value that the CEM chooses to represent a particular device model within the UTV network), and a serial number. The ULI Network Operating Center maintains a connection with each device. The NOC interface can display information about the device, control its registration, blacklist it, revoke provisioned credentials from it, and control which updates are sent to it.

EMBEDDED WEB SERVER

The Agent contains an optional embedded web server that is used for development and test. This is not for production use at this time. Please contact UpdateLogic for questions or additional information regarding this functionality.

CONSUMER PERSPECTIVE USE CASES

SOFTWARE UPDATE HAS BEEN STORED AND IS READY FOR INSTALLATION.

The Agent has scanned, found, and stored a compatible software update. If this software update arrived via Broadcast or Internet the user was not involved. If the update arrived via USB or direct download the user is already aware of the update. The System is notified that the download is complete. The System interprets the meta-data associated with the update and queries/notifies the user based on this data.

If the user is to be notified then the user is impacted as follows:

- The next time the TV is powered on (exits standby or cold boot), the user is notified that new software is available. The user must decide to either install now or install at their convenience in the future.
    - This screen will provide information about the update process such as how long the process takes, warnings about leaving the TV powered during the update, and that a TV power cycle will occur.
    - The user will be provided with details about the software update including the name of the software to be updated, the current and new versions of this software, and any other CEM specific details that can be extracted from the software update header.

- Should the user elect to install the update now, a confirmation screen is presented to the user. This screen should remind the user of the software update process. Confirmation will commence installation. Cancel, will return to the previous menu.
- Should the user elect to install at a later date, the user will be instructed on how to access the software update, via the TV menus, for installation at a more convenient time.

If the user is not to be notified of the software update, the user is impacted as follows:

- No user interaction occurs

MENU ENTRIES

The following menu entries could be added to the TV menu system

INSTALL PREVIOUSLY DOWNLOADED UPDATE

A software update that has been downloaded but not installed can be installed later by the user via this menu. This state is reached when:

a) A software update is downloaded without the "prompt user" attribute, and the CEM specific code does not override (always prompt) as such the update is stored but not installed.

b) A software update is downloaded and the user is prompted, but the user chooses to install at a later date.

This menu item should provide the user with information similar to that of the "prompt user" dialog box. See "Software update received" for more information on the content of this dialog box.

TECHNICAL SUPPORT

The UpdateTV Agent's nonvolatile storage contains a number of variables which could be presented to the user as diagnostic information. This information is of little use to the consumer, but is provided should a remote technical service representative need the data for remote diagnosis.

Main Help Screen

Electronic Serial Number of the device.

Current TV software version string

Hidden Information

ULI software version number

ULI module version

ULPK Index

Number of provisioned objects

Device Registered flag

Device in Factory Test Mode flag

Device in Field Test Mode flag

ULI Update Version String

Last Provision Status

Last Download Module Version

Last Download Type

Last Download Status

Last Download Time

Last Error

Last Error Time

Error Count

FACTORY PERSPECTIVE

The UpdateTV Agent supports a factory mode of operation in which the TV rapidly downloads a software image via broadcast mode on a known frequency and installs it.

FACTORY UPDATE

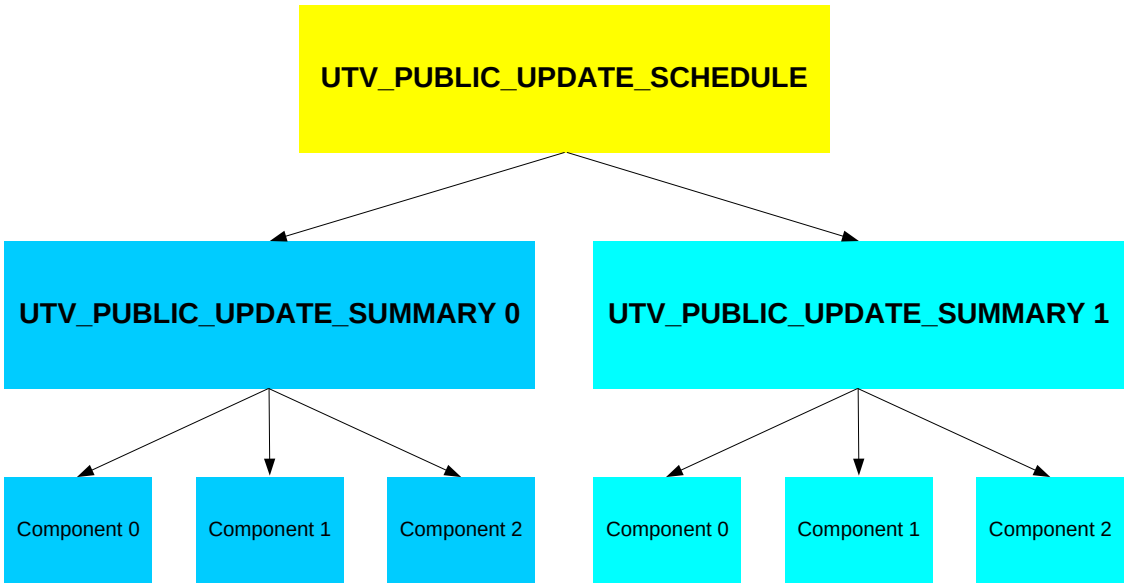
Using the UpdateTV SDK tools, the factory can be provided with a transport stream which includes all the information necessary for the UpdateTV Agent to detect, authenticate, download, and install a software updates for one or many production TVs simultaneously. The transport stream is spooled to a modulator which provides an ATSC compatible stream to the production line TVs via the in house RF test network. Once the RF is connected to the new TV, a menu item only accessible to the production environment is access.

- This menu presents the factory personnel with a single selection, “Factory Update Software”
- When pressed, the UpdateTV Agent is invoked in factory update mode. The UpdateTV Agent will”
    - Tune to the appropriate frequency
    - Identify and download and validate the software
    - Install the software
- During this time the factory personnel will be presented with a progress screen
- When complete the factory personnel will reboot the TV (as instructed)
- If an error occurs, the error will be displayed to the factory personnel

AGENT INTERFACE EXAMPLES

Actual example code demonstrating the cases described below can be found in the test code that comes with the Agent. In particular please look at the code in test/test-agent-project.c. The code in test-agent-project.c provides a wrapper for the functions described in this section. Calls to the functions in this section are actually found in project-XXXXX.c, but they shouldn't need to be modified. This section is provided as reference to understand how the Agent works and what interactions with it look like.

The main object that the Agent manages updates with is called the UTV_PUBLIC_SCHEDULE. There are 3 levels of nested structures within it. The root level is called the "Schedule". The Schedule points to multiple updates. The maximum number of updates it points to is controlled by the define `UTV_PLATFORM_MAX_UPDATES (platform-config.h)`. Even if the NOC has many compatible updates enabled if `UTV_PLATFORM_MAX_UPDATES` is set to one then the Agent will only see one of them at a time. Each update connects to the third level which is the component level. There can be up to `UTV_PLATFORM_MAX_COMPONENTS` in each update. The default value of this define is 100.



UTV_PUBLIC_COMPONENT_SUMMARY structures

The function `UTV_RESULT UtvPublicGetDownloadSchedule(UTV_BOOL bStoredOnly, UTV_BOOL bFactoryMode, UTV_PUBLIC_CALLBACK pCallback, UTV_UINT32 iToken, UTV_PUBLIC_SCHEDULE **pSchedule )` returns a pointer to the root schedule if one or more updates are available.

The application can then look at those updates and choose which one to accept. Typically `UTV_PLATFORM_MAX_UPDATES` is set to one which limits the Agent to dealing with one update at a

time which is the easiest option to manage. The NOC will present updates in module version order which enforces sequential delivery.

The function `UTV_RESULT UtvPublicApproveForDownload( UTV_PUBLIC_UPDATE_SUMMARY *pUpdate, UTV_UINT32 uiStore )` is used to approve download for all components in an update. If component by component approval is needed then the component structures under the update level can be exposed and each component can be approved for either store or install on a case by case basis. Note that the `uiStore` arg controls which store the update is downloaded into. Another new function called `UTV_RESULT UtvPublicSetUpdatePolicy( UTV_BOOL bInstallWithoutStore, UTV_BOOL bConvertOptional )` is used to control the behavior of `UtvPublicApproveForDownload()`. If the `bInstallWithoutStore` arg is set then all components will have their `bInstallWithoutStore` flags set when `UtvPublicApproveForDownload()` is called. The `bConvertOptional` converts optional updates to required updates.

Once the policy is established for the download then `UTV_RESULT UtvPublicDownloadComponents( UTV_PUBLIC_UPDATE_SUMMARY *pUpdate, UTV_BOOL bFactoryDecryptEnable, UTV_PUBLIC_CALLBACK pCallback, UTV_UINT32 iToken )` should be called to initiate the download. Note that it takes an update not a schedule as an argument. Only one update is processed at a time. If the `bInstallWithoutStore` arg was set when the call to `UtvPublicSetUpdatePolicy()` was made then each module of each component will be sent directly to `UtvPlatformInstallModuleData()` as it is downloaded. If the `bInstallWithoutStore` arg was not set then all components will be stored during download, but not installed. To install downloaded updates that have been stored a call to `UTV_RESULT UtvPublicInstallComponents( UTV_PUBLIC_UPDATE_SUMMARY *pUpdate, UTV_BOOL bFactoryDecryptEnable, UTV_PUBLIC_CALLBACK pCallback, UTV_UINT32 iToken )` can be made.

UNDERSTANDING THE COMPONENT MANIFEST

The component manifest is an encrypted file generated by Publisher that contains the as-shipped ULI identity of the updateable software in the device. When the device boots for the first time the Agent detects that the component manifest isn't present in the "active component manifest" directory (specified by the UTV_PLATFORM_ACTIVE_COMP_MANIFEST_FNAME define) and copies it from the "backup component manifest" directory (specified by the UTV_PLATFORM_BACKUP_COMP_MANIFEST_FNAME define). When an update takes place the Agent copies over the active component manifest with the "update component manifest" that arrives with the update during the download and install process.

A sample Publisher script file called test/cm.txt is supplied with the distribution of the Agent source. It contains the Publisher directives used to create a component manifest. The distribution comes with a backup component manifest (read-only/component.manifest) pre-built from that script file. Cm.txt is provided so you can see how it's created.

Likewise the distribution comes with a Publisher script file called test/cmf.txt. This script was used to create the file read-only/component.manifest.factory that is also distributed with an Agent release. This

file contains a “factory component manifest” (whose location is specified by the UTV_PLATFORM_FACTORY_COMP_MANIFEST_FNAME define). The factory component manifest contains an attribute that identifies the device as being in “factory mode”. This is done by specifying the –factory_test directive in the script file. In this mode it will only be able to receive updates that have likewise been published using a script that contains a –factory_test directive. When the TV switches from MP mode to factory mode the factory component manifest is coped over the active component manifest.

The component manifest specifies a number of identifying attributes of the update it represents. This “update” includes the software that is initially shipped from the factory even though it may have been delivered by a gang programmer and not the UpdateTV process.

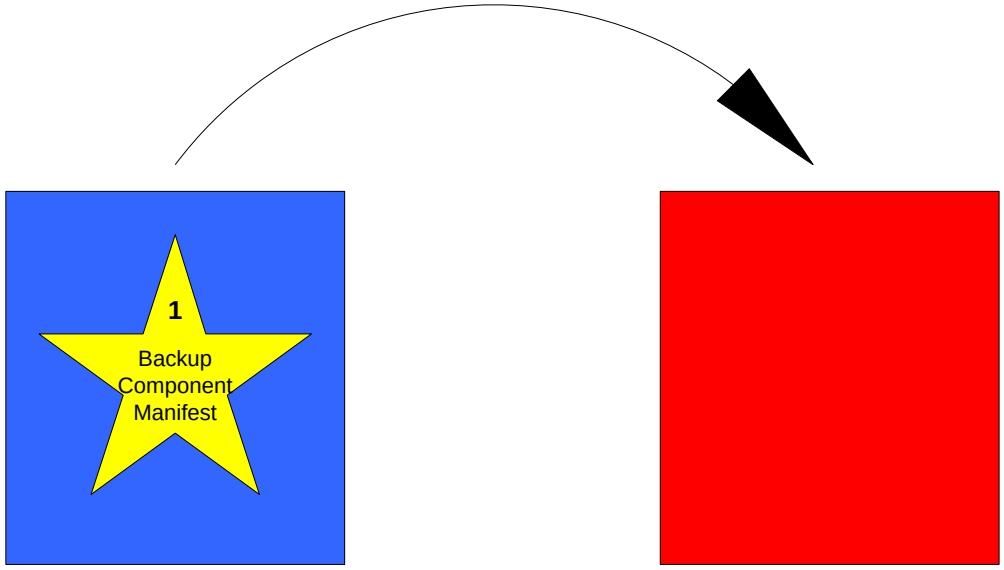
These identifying attributes include *update-level* (global) parameters that apply to all components in the update. The directives used to specify these attributes are described in more detail in the *UpdateLogic Publisher* document. The update-level attributes include:

- Software version number for the update as a whole. This is an integer that should be allocated for each update created. It’s specified by the –version XX directive. You will have to create a unique version number on the NOC for each update you create (using Settings->Software Versions->Add Software Version) and assign that number to the update using this directive.
- Module version. The module version is automatically maintained by the NOC and supplied to updates as part of the Publishing process *except* in the case of a backup component manifest in which case the module version is specified as zero using the –mv 0 directive.
- Any special text directives needed for the update. These may include a version string, short release notes, etc. These
- Although Publisher enforces the inclusion of the compatibility signaling directives such as –for_hardware_models X X and –for_software_versions Y Y they have no application when building a component manifest because the “update” is already installed. You may therefore place any values you wish for those two directives.

Each component in the installed software should also be defined in the component manifest. These component-level attributes include:

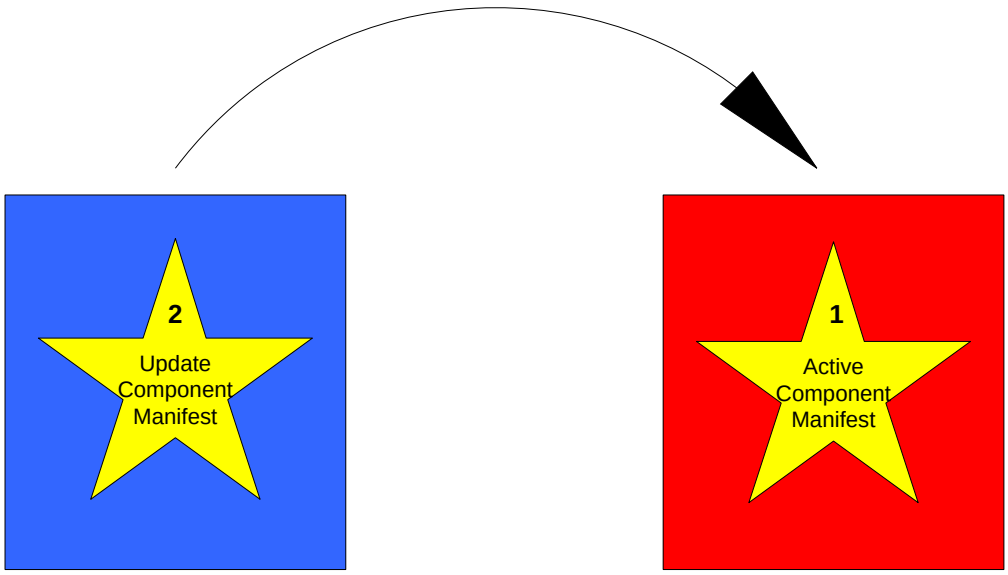
- A definition of the names of each of the components. This is done using the –name “component name” directive from inside of a curly bracket pair which indicates the definition of a component.
- A definition of the version of each of the components. This is done using the –version XX directive from inside of a curly bracket pair which indicates the definition of a component.
- A backup component manifest doesn’t contain actual component payload. It’s just a directory of components that were flashed into the device by a factory automation process. Therefore dummy data is specified in the component definitions in a backup component manifest using the –test 0x100 directive.
- A definition of the partition attribute of the component may be specified using the –partition directive if the component is a partition. Omitting the –partition directive tells the Agent that the component is a file instead of a partition.
- Although Publisher enforces the inclusion of a software version compatibility signaling directive (–for_software_versions Y Y) at the component-level it has no application when building a component manifest because the “update” is already installed. You may therefore place any values you wish for that directive.

COMPONENT MANIFEST UPDATE DIAGRAMS



Backup/Read-Only **Active Read/Write**

On first boot the backup component manifest is copied from a read-only partition to the read-write active component manifest directory.



Update/Read-Only **Active Read/Write**

When an update is successfully downloaded the active component manifest is overwritten by the update component manifest.

SECURETV ONLY PORTS

If you're only porting SecureTV then many of steps described above do not apply to your Agent integration effort. The steps above that are for UpdateTV only are marked in red. SecureTV must still be told the version of the firmware that it is integrated into in order to control cert access that may be gated on version. The SecureTV version is stored in an encrypted file called the "component manifest" as described in the previous section. Normally SecureTV is responsible for updating the component manifest. In the case of SecureTV only ports the component manifest needs to be updated every time the application containing SecureTV is updated. If that application doesn't change then the component manifest doesn't need to change either. This is done to prevent certs from being provisioned to devices that have known security issues.

The backup and factory mode component manifests don't apply to a SecureTV only port. There is only one read-only component manifest.

SUPPORTTV ONLY PORTS

SupportTV is an optional feature set of the Agent that allows the device to engage in live remote support sessions whereby authorized technical support personnel can access the device in real-time over the Internet via a web browser. SupportTV capability is disabled by default and must be explicitly enabled in the Agent source code as desired (see the Quick Start section later in this document).

Even integration projects that do not currently plan to utilize SupportTV in production can enable it for development evaluation. The integration effort required to establish a basic live support session is low. Please contact UpdateLogic to request that your model group be authorized for use with SupportTV in development environments.

SupportTV integration builds upon the base integration efforts described earlier in this document. The device must already be integrated to the extent that it can register with the UpdateLogic NOC as described in the Quick Start Porting Instructions section earlier in this document. SupportTV is not dependent upon the functionality UpdateTV or SecureTV, and so the device does not already have to be capable of software or security credential updates.

QUICK START

The following section provides "quick start" instructions for enabling the base SupportTV functionality of the Agent source for your device. This section builds on the instructions found in the Quick Start Porting Instructions section earlier in this document..

BUILDING A TEST VERSION OF THE AGENT WITH SUPPORTTV

The default Makefile contains the necessary configurations to enable the base capabilities of SupportTV. The first step is to remove the comment character (#) in front of the "PLATFORM_LIVE_SUPPORT=1" define. Defining PLATFORM_LIVE_SUPPORT in the Makefile also enables the GCC compile option "*-std=c99*".

The base SupportTV functionality has some additional library dependencies: *libm*, *libz* and *uuid*. One must verify these libraries are available to successfully build the test version of the Agent.

Once the Makefile changes have been made and the additional library dependencies have been made available the test version of the Agent can be built using the instructions found in the Building a Test Version of the Agent instructions.

EXERCISING TEST-AGENT-PROJECT WITH SUPPORTTV

First follow the instructions for Exercising test-agent-project to validate the non-SupportTV Agent capabilities. Once that is completed the first SupportTV command to exercise is 't'. This will test the connection with the NOC using the SupportTV communication path. You will see something like the following when you hit 't':

```
UTV Wed Oct 27 14:31:32 (main) Echo from Agent
```

The next functionalities to test are the initiation and termination of a SupportTV session. Use the 'l' command to both initiate and terminate a session. You will see something like the following when you hit 'l' the first time:

```
UTV Wed Oct 27 14:54:32 (main) Live Support session (123-456-789) initiated
```

The number inside the parentheses is the session code for the SupportTV session.

You will see something like the following when you hit 'l' for the second time:

```
UTV Wed Oct 27 15:08:33 (main) Live Support session (123-456-789) terminated
```

The sequence of initiating and terminating the SupportTV session can be repeated. Each time you will see a different session code displayed in the messages.

OPERATING A LIVE SUPPORT SESSION

With the device in a SupportTV session as triggered by the 'l' command of test-agent-project, one can connect to the device using a web browser. See section Exercising the SupportTV Client later in this document for information on how to connect to your device using its session code.

SUPPORTTV PLATFORM ADAPTATION FUNCTIONS

This section describes the platform adaptation functions for SupportTV that are contained in the file project-XXXX.c. These functions *must* be adapted by the integrator to perform platform-specific versions of the functions they represent. The operational goal and placeholder function of each of these entry points is described.

SESSION MANAGEMENT ADAPTATION FUNCTIONS

`void UtvCEMLiveSupportInitialize ( void )`

- This function is used to handle platform specific initialization for SupportTV.
- The placeholder code registers sample command handlers for all ULI commands, a sample custom diagnostic command and a sample tool command.
- Replace the body of this function with the registration of command handlers supported by the device.

`UTV_RESULT UtvCEMGetDeviceInfo ( cJSON * pDeviceInfo )`

- This function is used to handle requests for device information.
- Example implementation gathers device information and adds the results to the cJSON object.
- Replace the body of this function with a device specific implementation that retrieves the device information and adds the results to the cJSON object. The example implementation contains the expected fields; additional device specific fields can be added.

`UTV_RESULT UtvCEMGetNetworkInfo ( cJSON * pNetworkInfo )`

- This function is used to handle requests for device information.
- Example implementation gathers device information and adds the results to the cJSON object.
- Review the implementation of the platform adaptation function, UtvPlatformGetNetworkInfo, and if necessary replace the body of this function with a device specific implementation that retrieves the device information and adds the results to the cJSON object.

`UTV_RESULT UtvCEMGetNetworkDiagnostics ( cJSON * pNetworkDiagnostics )`

- This function is used to handle requests for device information.
- Example implementation gathers device information and adds the results to the cJSON object.
- Replace the body of this function with a device specific implementation that executes the network diagnostic tests. The example implementation contains the expected tests and expected fields; additional device specific tests and fields can be added.

```
UTV_RESULT UtvPlatformGetNetworkInfo ( void * pVoidResponseData )
```

- This function is used to handle requests for network information.
- The reference implementation retrieves network information using Linux networking functionality for the active/enabled interfaces.
- If the reference implementation is not supported by the platform or can be enhanced with device specific processing the body of this function can be replaced. As an alternative the implementation of UtvCEMGetNetworkInfo can also be replaced and this function can be ignored.

OTHER ADAPTATION FUNCTIONS

There are many other functions in the project-xxxx.c file. These functions should not have to be adapted to be used on your platform. They're provided to improve understanding and create an at-a-glance view of the project-related functions that are operating in your integration.

SUPPORTTV COMMAND HANDLERS

The SupportTV implementation consists of platform adaptation functions, described in the above sections, and registered command handlers. The command handlers utilize a Third Party library, cJSON, to implement a C version of Java Script Object Notation (JSON). The cJSON structures provide the framework for the inputs and outputs to the command handlers.

All command handlers have the same format and are registered by calling the register method, UtvLiveSupportTaskRegisterCommandHandler.

```
void UtvLiveSupportTaskRegisterCommandHandler ( UTV_BYTE * command_name, UTV_BYTE * command_description_file_path, UTV_RESULT (*command_handler)( cJSON * inputs, cJSON * outputs ) )
```

The commands handled by the command handlers come in two forms, ULI commands and CEM commands. ULI commands are expected to be handled by all devices in some form. CEM commands provide an extensible framework for integrators to add additional device specific functionality.

The command names specified in the registration requests are prefixed with an OUI (6 character hexadecimal string) followed by the name of the command. With this specification the ULI commands are prefixed with "FFFFFF". For example, the command to handle the retrieval of network configuration is "FFFFFF_LIVE_SUPPORT_GET_NETWORK_CONFIG".

The command description specified in the registration requests is JSON formatted string containing the following information:

- TYPE – Type of the command, either TOOL_COMMANDS or DIAGNOSTIC_COMMANDS.
- ID – Identifier of the command, this is the OUI prefixed value to uniquely identify this command from others.
- NAME – User friendly name of the command.
- DESCRIPTION – Detailed information about the command.
- ORDER
- PERMISSIONS
- INPUTS – Detailed information about the input values for this command.
    - ID
    - NAME
    - DESCRIPTION
    - TYPE
    - DEFAULT
    - VALUES
    - REQUIRED
    - VISIBLE
- OUTPUTS – Detailed information about the output values from this command.
    - ID
    - NAME
    - DESCRIPTION
    - ORDER
    - TYPE

Examples of the device command descriptions are found in project/project-XXXX.h.

The command handler specified in the registration requests is the function to call when the command is requested by SupportTV. The input and output of the handlers are cJSON objects, the contents of which are specific to each command.

CONTROL COMMANDS

This section describes commands that are used by SupportTV to remotely control aspects of the device.

FFFFFF_LIVE_SUPPORT_TV_REMOTE_BUTTON

- Command handler called by the ULI SupportTV code to provide one or more TV Remote Buttons to be processed by the device.
- Example implementation (s_LiveSupportProcessTvRemoteButton) loops through the TV Remote Button inputs.
- Replace the processing of the individual button within the loop to perform device specific conversion and processing of the button.

DIAGNOSTIC COMMANDS

This section describes commands that are used by SupportTV to retrieve diagnostic information from the device.

FFFFFF_LIVE_SUPPORT_GET_DEVICE_INFO

- Command handler called by the ULI SupportTV code to retrieve device specific information from the device.
- Example implementation (s_LiveSupportGetDeviceInfo) gathers device information and adds the results to the output data.
- Replace the body of this command handler function with a device specific implementation that retrieves the device information and adds the results to the output data. The example implementation contains the expected fields; additional device specific fields can be added.

FFFFFF_LIVE_SUPPORT_GET_NETWORK_INFO

- Command handler called by the ULI SupportTV code to retrieve network information from the device.
- Example implementation (s_LiveSupportGetNetworkInfo) gathers network information and adds the following values to the cJSON output object using UtvPlatformGetNetworkInfo.
- Replace the body of this command handler function with a device specific implementation that retrieves the network information and adds the results to the output data. The example implementation contains the expected fields; additional device specific fields can be added.

FFFFFF_LIVE_SUPPORT_HARDWARE_DIAG_QUICK

- Command handler called by the ULI SupportTV code to request the device execute a quick hardware diagnostic test and return the results.
- Example implementation (s_LiveSupportHardwareDiagQuick) returns hard-coded sample data in JSON format.
- Replace the body of this command handler function with a device specific implementation that executes a quick hardware diagnostic check and adds the results to the output data.

FFFFFF_LIVE_SUPPORT_HARDWARE_DIAG_FULL

- Command handler called by the ULI SupportTV code request the device execute a full hardware diagnostic test and return the results.
- Example implementation (s_LiveSupportHardwareDiagFull) returns hard-coded sample data in JSON format.
- Replace the body of this command handler function with a device specific implementation that executes a full hardware diagnostic check and adds the results to the output data.

FFFFFF_LIVE_SUPPORT_CHECK_INPUTS

- Command handler called by the ULI SupportTV code to check the status of the device input ports (i.e. TV, AVI, HDMI, RGB, etc.).
- Example implementation (s_LiveSupportCheckInputs) returns hard-coded sample data in JSON format.
- Replace the body of this command handler function with a device specific implementation that checks the status of the device input ports and adds the results to the output data. The results should include a list of the inputs available and a true/false indicator for their connected status.

FFFFFF_LIVE_SUPPORT_GET_PARTS_LIST

- Command handler called by the ULI SupportTV code to retrieve a list of the parts that make up the device. This information can include additional information such as part revision or manufacturer.
- Example implementation (s_LiveSupportGetPartsList) returns hard-coded sample data in JSON format.
- Replace the body of this command handler function with a device specific implementation that retrieves the parts list for the device and adds the results to the output data.

FFFFFF_LIVE_SUPPORT_GET_SYSTEM_STATISTICS

- Command handler called by the ULI SupportTV code to retrieve network configuration information from the device.
- Example implementation (s_LiveSupportGetSystemStatistics) gathers statistical information for the system and adds the following items to the cJSON output object using UtvPlatformGetSystemStats.
- Replace the body of this command handler function with a device specific implementation that retrieves statistical information for the device and adds the results to the output data. The example implementation contains the expected fields; additional device specific fields can be added.

TOOL COMMANDS

This section describes commands that are used by SupportTV to alter the configuration or characteristics of the device.

FFFFFF_LIVE_SUPPORT_RESET_FACTORY_DEFAULTS

- Command handler called by the ULI SupportTV code to request that the device reset to factory defaults.
- Example implementation (s_LiveSupportResetFactoryDefaults) does nothing.
- Replace the body of this command handler function with a device specific implementation that restores the factory settings of the device.

FFFFFF_LIVE_SUPPORT_SET_ASPECT_RATIO

- Command handler called by the ULI SupportTV code to modify the aspect ratio of the device.
- Example implementation (s_LiveSupportSetAspectRatio) retrieves the new aspect ratio to apply from the input information.
- Replace the body of this command handler function with a device specific implementation that adjusts the aspect ratio using the command inputs. The command inputs include a ratio in the form of WIDTH by HEIGHT (“WxH”) or one of the special values “Zoom” or “Stretch”.

FFFFFF_LIVE_SUPPORT_APPLY_VIDEO_PATTERN

- Command handler called by the ULI SupportTV code used to request that the device display a specific video pattern.

- Example implementation (s_LiveSupportApplyVideoPattern) retrieves the video pattern to apply from the input information.
- Replace the body of this command handler function with a device specific implementation that applies the video pattern from the command inputs to the device. The command inputs include the type of pattern to apply “SIMPLE” or “NONE” and the pattern value specified as hexadecimal ARGB string (i.e. “FFFF0000” for red).

FFFFFF_LIVE_SUPPORT_GET_DEVICE_CONFIG

- Command handler called by the ULI SupportTV code to retrieve device configuration.
- Example implementation (s_LiveSupportGetDeviceConfig) reads the contents of the */etc/services* file, Base64 encodes the data and return it in the output data.
- Replace the body of this command handler function with a device specific implementation that retrieves the device configuration information. The information must be Base64 encoded and added to the output data.

FFFFFF_LIVE_SUPPORT_SET_DEVICE_CONFIG

- Command handler called by the ULI SupportTV code to update the device configuration.
- Example implementation (s_LiveSupportSetDeviceConfig) Base64 decodes the input information and writes the data to the *tmpConfig* file.
- Replace the body of this command handler function with a device specific implementation that Base64 decodes the input data, validates the decoded data and writes the data to the device configuration.

CUSTOM DIAGNOSTIC COMMANDS

The SupportTV command framework provides an extensible mechanism for device manufactures to add additional diagnostic capabilities.

The Agent code provided implements a sample diagnostic command with the identifier FFFFFF_CUSTOM_DIAGNOSTIC_SAMPLE. The sample implementation (s_LiveSupportCustomDiagnosticSample) returns HTML formatted text.

Custom commands added within the device are retrieved by the UpdateLogic NOC at run-time and made available in the SupportTV client interface to technicians. Custom commands do not have to be pre-registered in the UpdateLogic NOC at design-time.

CUSTOM TOOL COMMANDS

The SupportTV command framework provides an extensible mechanism for device manufactures to add additional diagnostic capabilities.

The Agent code provided implements a sample tool command with the identifier FFFFFF_CUSTOM_TOOL_SAMPLE. The sample implementation (s_LiveSupportCustomToolSample) retrieves various types of data from the input information and returns simple text containing the command handler method name.

Custom commands added within the device are retrieved by the UpdateLogic NOC at run-time and made available in the SupportTV client interface to technicians. Custom commands do not have to be pre-registered in the UpdateLogic NOC at design-time.

ADDING VIDEO CAPABILITIES

The default Makefile contains the necessary configurations to enable the video capabilities of SupportTV. The first step is to remove the comment character (#) in front of the "PLATFORM_LIVE_SUPPORT_VIDEO=1".

The video capabilities of the SupportTV functionality have additional library dependencies. See the *NetReady Agent Integration Requirements* document for a table listing the library dependencies. One must cross-reference this list to an existing device platform to identify the new library dependencies required by SupportTV.

Once the Makefile changes have been made and the additional library dependencies are available the test version of the Agent can be built using the instructions found in the Building a Test Version of the Agent.

INSTALLING ADDITIONAL PACKAGES

Many of the library dependencies described above are available for many Linux distributions. Installing these using the default Linux distribution is a quick method to resolving the missing dependencies but may install more features than are required by SupportTV.

Example of installing on Ubuntu

- sudo apt-get install gstreamer0.10-plugins-good
- sudo apt-get install libgstreamer0.10-dev
- sudo apt-get install libgstreamer-plugins-base0.10-dev
- sudo apt-get install liboil0.3

In addition to installing the packages there are two other steps required.

1. The liboil package installation does not automatically create liboil-0.3.so. To resolve this execute "sudo ln –s /usr/lib/liboil-0.3.so.0 /usr/lib/liboil-0.3.so"
2. The GStreamer package installation does not automatically add itself to the library path. To resolve this do the following items.
    a. Edit /etc/ld.so.conf
    b. Add "/usr/lib/gstreamer-0.10"
    c. sudo ldconfig

For information on installing or building the library dependencies with the minimal dependencies contact UpdateLogic.

VIDEO CONTROL ADAPTATION FUNCTIONS

UTV_BOOL UtvPlatformLiveSupportSecureMediaStream(void)

- Called by the ULI SupportTV video code to determine if the device supports securing the media stream
- Example implementation returns UTV_TRUE if UTV_LIVE_SUPPORT_VIDEO_SECURE is defined and returns UTV_FALSE otherwise
- Securing the media stream requires the UpdateLogic changes to the GStreamer TCP base plug-in; contact UpdateLogic for additional information regarding these changes

UTV_RESULT UtvPlatformLiveSupportVideoGetLibraryPath(UTV_BYTE * pubBuffer, UTV_UINT32 uiBufferSize)

- Called by the ULI SupportTV video code to retrieve the path to the GStreamer libraries (i.e. /usr/lib/gstreamer-0.10)
- Example implementation returns "/usr/lib/gstreamer-0.10" which is the standard installation path for GStreamer libraries.

UTV_RESULT UtvPlatformLiveSupportVideoGetVideoFrameSpecs(UTV_INT32 * piWidth, UTV_INT32 * piHeight, UTV_INT32 * piPitch)

- Called by the ULI SupportTV video code to retrieve the dimensions (width, height and pitch) of the video frame. These values are used to calculate the size of the buffer used in the UtvPlatformLiveSupportVideoFillFrameBuffer method.
- Example implementation returns dimensions of 960 x 1080 x 4 for width, height and pitch respectively.

UTV_RESULT UtvPlatformLiveSupportVideoFillVideoFrameBuffer(UTV_BYTE * pubBuffer, UTV_UINT32 uiBufferSize)

- Called by the ULI SupportTV video code to retrieve a video frame buffer.
- The frame rate established when a SupportTV client enables video establishes the rate at which this method is called. The default frame rate is two frames per second which results in this function being called approximately twice per second.
- Example implementation generates two solid colored buffers (both shades of gray) switching every 5th call of the function.

GSTREAMER CONFIGURATION

The video capabilities of SupportTV are implemented using a Third Party package, GStreamer. There are required features from the core, base plug-ins and good plug-ins. The information in each of the sections below describes the configuration options enabled and the libraries used by the Agent.

CORE

Configure Options

- libtool-lock – Enable locking (supports parallel builds)

- gst-debug – Enable debugging subsystem
- trace – Enable tracing subsystem
- alloc-trace – Enable allocation tracing
- net – Enable network distribution
- plugin – Enable plug-ins

Libraries (paths are relative from the root of the ***gstreamer*** source tree)

- ./gst/.libs/libgstreamer-0.10.so*
- ./libs/gst/base/.libs/libgstbase-0.10.so*
- ./libs/gst/check/.libs/libgstcheck-0.10.so*
- ./libs/gst/controller/.libs/libgstcontroller-0.10.so*
- ./libs/gst/dataprotocol/.libs/libgstdataprotocol-0.10.so*
- ./libs/gst/net/.libs/libgstnet-0.10.so*
- ./plugins/elements/.libs/libgstcoreelements.so*

BASE PLUG-INS

Configure Options

- libtool-lock – Enable locking (supports parallel builds)
- rpath – Hardcode runtime library paths
- external – Enable building of plug-ins with external dependencies
- app – Enable “app” plug-in
- ffmpegcolorspace – Enable “ffmpegcolorspace” plug-in
- tcp – Enable “tcp” plug-in
- videoscale – Enable “videoscale” plug-in

Libraries (paths are relative from the root of the ***gst-plugins-base*** source tree)

- ./gst/app/.libs/libgstapp.so*
- ./gst-libs/gst/app/.libs/libgstapp-0.10.so*
- ./gst-libs/gst/video/.libs/libgstvideo-0.10.so*
- ./gst/ffmpegcolorspace/.libs/libgstffmpegcolorspace.so*
- ./gst/tcp/.libs/libgsttcp.so*
- ./gst/videoscale/.libs/libgstvideoscale.so*

GOOD PLUG-INS

Configure Options

- libtool-lock – Enable locking (supports parallel builds)
- rpath – Hardcode runtime library paths
- external – Enable building of plug-ins with external dependencies
- jpeg – Enable “jpeg” library

Libraries (paths are relative from the root of the ***gst-plugins-good*** source tree)

- .ext/jpeg/.libs/libgstjpeg.so*

EXERCISING THE SUPPORTTV CLIENT

CONNECTING TO THE DEVICE

The first step in exercising the SupportTV client is starting test-agent-project and initiating a SupportTV session (see Exercising test-agent-project with SupportTV).

Once a session is initiated, open a web browser (such as Microsoft Internet Explorer) and direct it to a NetReady SMS site.

- Pre-release SupportTV development using Device Agent 3.0.0 uses the NetReady Demo NOC:
  - http://live.demo.nocs.updatelogic.com
- Pre-production development using Device Agent 3.0.1 or later use the NetReady ExtDev NOC:
  - https://live.extdev.updatelogic.com
- Production devices use the NetReady Production NOC:
  - https://live.support.updatelogic.com

Provide credentials as prompted to log into the NOC. See UpdateLogic for appropriate credentials.

After login you are presented with the SupportTV session connection page. Enter the session code displayed from test-agent-project into the edit box and press the “Connect” button.

WORKING WITH THE DEVICE

The SupportTV client interface is fairly straight-forward. Explore the tabs (Overview, Control, Diagnostics, Tools, etc…) as desired. Use the “Disconnect” link in the upper-right corner of the interface to end the live support session.

If the SupportTV adaption functions and commands have not yet been customized then some information displayed and the results of running diagnostics and tools will be based on the default implementation and not be fully accurate for the device.

If the streaming video capability of SupportTV has not yet been enabled then the video feed on the Control tab will not display any video. If streaming video is enabled but not fully integrated, the default behavior will result in alternating shades of gray periodically.

Note that the test-agent-project output will display relevant activity as the facilities of the SupportTV client interface are exercised.