7.8.2 Typedef Documentation

7.8.2.1 typedef uint32(∗ svcs_statusNotifyCb)(svcs_statusEvent event)

Defines a services status-notification callback function.

This function is called asynchronously to status notifications for DTA mode, system time, and EAS events.

Parameters

notification	is key notification data

Return values

TRUE	if the notification is successful
FALSE	if the operation fails

7.8.2.2 typedef uint8 svcs_statusVcInfoAttr

7.8.3 Enumeration Type Documentation

7.8.3.1 enum svcs_statusCCI

Defines copy-control information.

Enumerator:

STATUS_CCI_COPY_FREELY	Indicates that the content is not copy-protected.
STATUS_CCI_COPY_NO_MORE	Is essentially the same as `STATUS_CCI_COPY_ONCE`.
STATUS_CCI_COPY_ONCE	Indicates that a single copy of the recording can be made, but no more.
STATUS_CCI_COPY_NEVER	Indicates that the content can be recorded and viewed for 90 minutes after transmission, and is not transferable.
STATUS_CCI_DIGITAL_COPY_ONCE	Indicates that the content is copyable once for digital output, but will have a Macrovision 7-Day Unlimited restriction on analog copies.
STATUS_CCI_DIGITAL_COPY_NEVER	Indicates that the content is not copyable for digital (delete after 90 minutes), but will have a Macrovision 7 Day Unlimited restriction on analog copies.

7.8.3.2 enum svcs_statusDtaMode

Defines the DTA modes.

Enumerator:

DTA_MODE_HUNT_INITIAL	Hunt, Initial mode (under 3 seconds)
DTA_MODE_HUNT_PROGRESS	Hunt, Progress mode (over 3 seconds, 0-1 iterations complete)
DTA_MODE_HUNT_DSI	Hunt, Delayed Service Interruption mode (2 iterations complete)
DTA_MODE_PEND_INIT_INITIAL	Pending Init, Initial mode.
DTA_MODE_PEND_INIT_AS	Pending Init, Activation Support mode.

TUNER_DISCONNECTED	The disconnection succeeded.
TUNER_CONNECTION_FAILED	The connection failed.

7.9.3.3 enum svcs_tunerNotifyEvent

Defines the types of tuner notification events.

Enumerator:

TUNER_CONNECTION_STATUS	A connection status event.
TUNER_PRESENTATION_STATUS	A presentation status event.

7.9.3.4 enum svcs_tunerPresStatus

Defines the tuner audio/video presentation status types.

Enumerator:

TUNER_PRES_NORMAL	Audio/video presentation is normal (A/V presented)
TUNER_PRES_LOCKED	Audio/video presentation is locked (no A/V presented)
TUNER_PRES_NOT_AUTH	Audio/video presentation is non-authorized (no A/V presented)
TUNER_PRES_FAILED	Audio/video presentation has failed (no A/V presented)

7.9.4 Function Documentation

7.9.4.1 void svcs_tunerClearVcnLocks ( )

Clears all channel locks.

7.9.4.2 uint16 svcs_tunerConnect (svcs_tunerConnectReq *req*)

Connects (tunes) the tuner to a specific service.

After making the transport connection, this call automatically selects the audio and video tracks for immediate presentation to the user. The video track selection is always the default video track. The audio track selection is based on the ISO 639 language code of the last audio track that the user manually selected (see svcs_tunerSetSelectedAudioTrack() for details). If no audio track with that language code exists on the current service, the default audio track is selected.

Note

The prior connection, if any, is automatically (and silently) disconnected.

Parameters

req	is a pointer to the tuner connection-request structure

Return values

SUCCESS	if the operation is successful
FAILURE	if an error occurs

Session Setup and Teardown Message Flows

This section provides several examples of message flow for session setup and teardown, from the HME app through the SCM to the DNCS/SRM or VOD Pump, and back:

- Figure 2 shows a session setup with immediate billing.
- Figure 3 shows a session setup with billing delayed until the HME confirms that it is successfully decoding the video stream
- Figure 4 shows four options for session teardown.

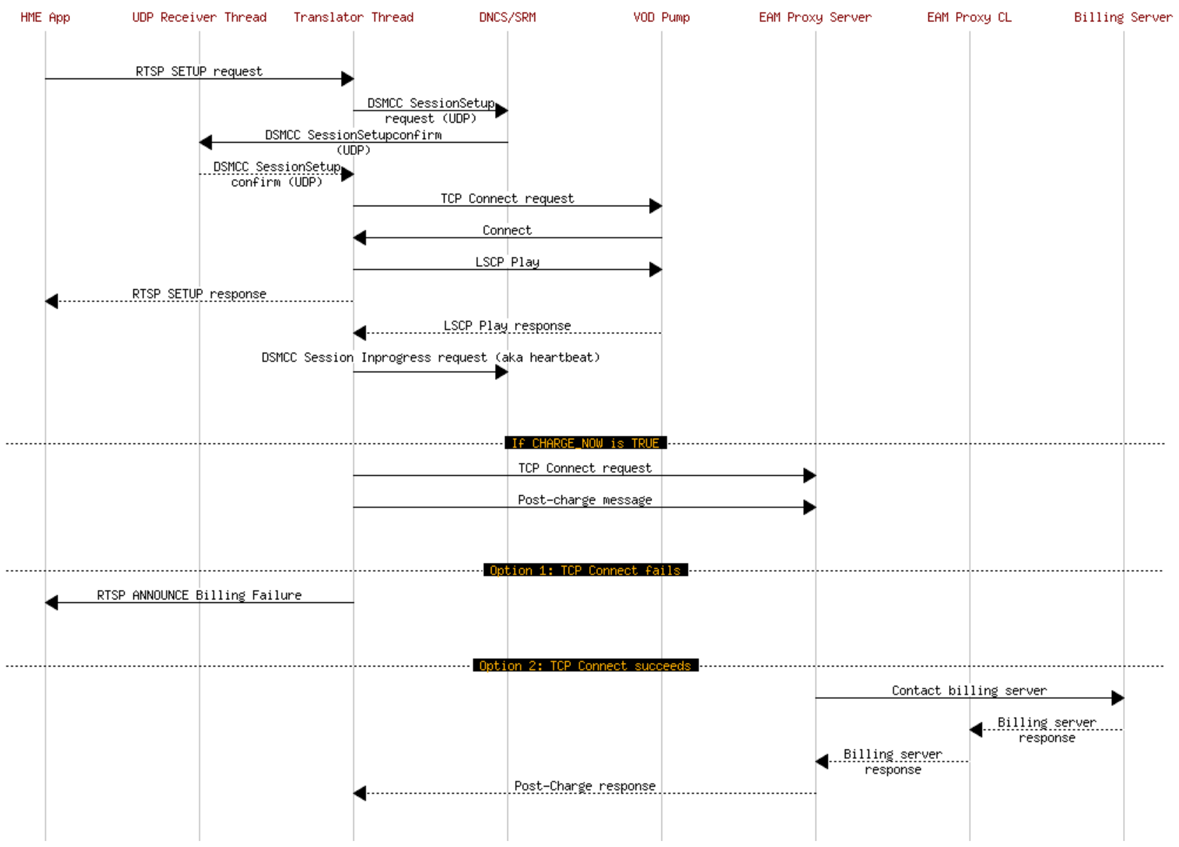


Figure 2 – Session Setup (charge immediately)

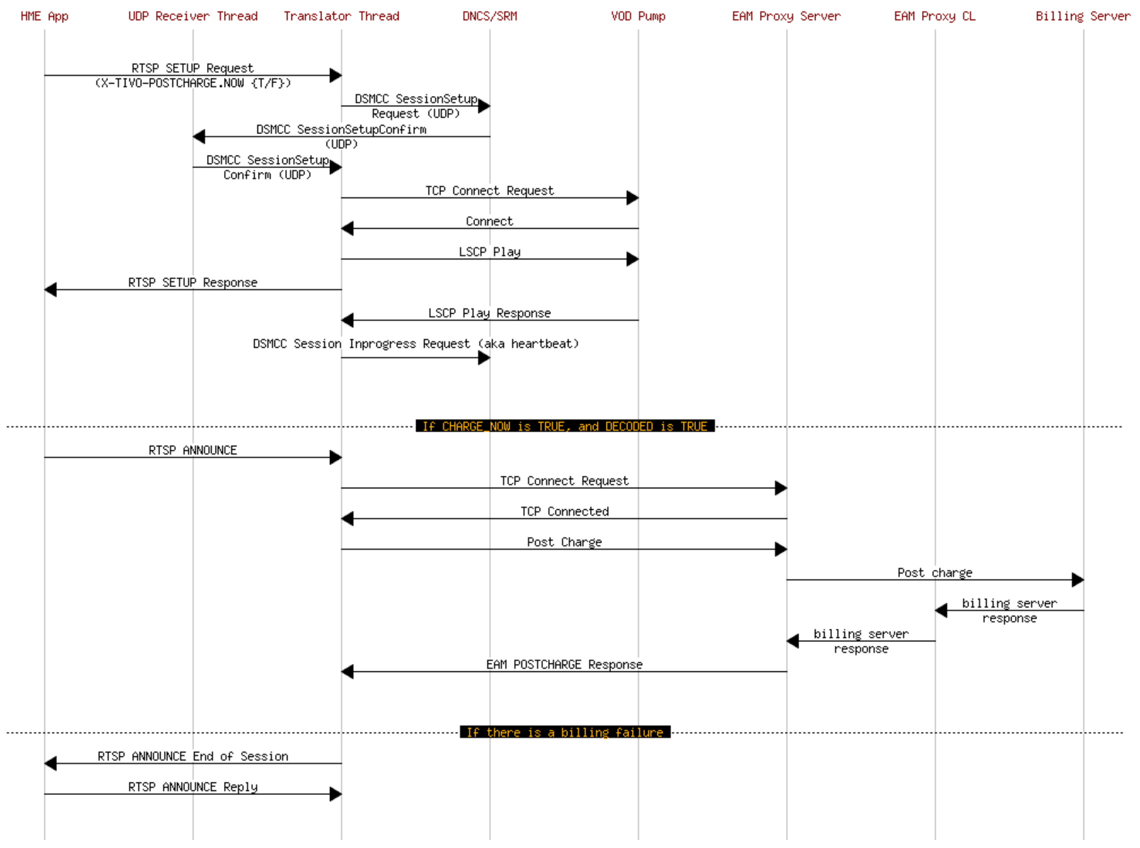


Figure 3 – Session Setup (delayed charge)

Session Setup Request

The Message flow starts with a TCP connection request from the HME app to the SCM on the RTSP port. The SCM creates a translator thread and passes it the connection details, allowing the thread to establish a TCP RTSP connection with the HME app.

The HME app then sends the RTSP session setup request to the translator thread. The thread creates a DSMCC session setup request message using the data in the RTSP session setup request message, data retrieved from the DB, and data from the Proxy API.

Database queries provide the conversion from a TiVo body ID and asset ID to an ARRIS asset ID. In addition, the database query returns additional ARRIS-specific asset information required to form the DSMCC session setup private data. The **getArrisAssetInfo()** API call provides this functionality.

NOTE: In addition to a unique name, the TiVo body ID contains the set-top box's 4-character Node Group ID and 12-character MAC address. In the following example, the Node Group ID is **000A**, and the MAC address is **001BD77F7F7D**:
bd.tomsstuff000A001BD77F7F7D

The Proxy APIs provide methods to acquire set-top specific information, such as Service Group.

The DSMCC session setup message is then sent to the DNCS/SRM over the translator thread's UDP communication port.

At this point, the translator thread can receive additional messages from the RTSP port.

[redacted]

Appendix A: API documentation

This appendix provides API documentation for inter-component communications, such as database access and translation between [redacted] and ARRIS data structures.

Database access methods

Methods in this section provide database-access functionality, such as invoking a data lookup to translate a [redacted] Offer ID into an ARRIS Asset ID, and retrieving the metadata associated with an Asset ID.

long getArrisAssetId(long [redacted])

The stream control proxy uses the **getArrisAssetId()** method to supply a [redacted] to the metadata adapter. The metadata adapter looks up the Offer ID, and provides the corresponding ARRIS Asset ID.

ArrisAssetId contains a CMM ODA metadata set ID (the AssetCollection) and a CMM Product ID (the AssetInfo). The AssetCollection element is an instance of **ModCharacteristicsInfo::modElementId**, 4-byte format, and the AssetInfo is an instance of **AssetInfo::modElementId**, 4-byte format.

String getArrisAssetInfo(long [redacted])

The stream control proxy uses the **getArrisAssetInfo()** method to supply a [redacted] Offer ID to the metadata adapter. The translator thread passes the [redacted] to the metadata adapter, which returns an **ArrisAppData** string.

[redacted] is the unique [redacted] identifier of the given offering. contains **AssetCollection** and **AssetId**, where the AssetCollection is the CMM ODA metadata set ID (**ModCharacteristicsInfo::modElementId**, 4-byte format), and the AssetId is the CMM Product ID (**AssetInfo::modElementId**, 4-byte format).

ArrisAppData is a string containing the following semicolon-delimited ARRIS Proprietary and Confidential properties:

oda is the name of the On Demand Application (ODA) making the request. The ARRIS back office applies any required logic based on the value of this parameter, allowing different clients to be accommodated.
This property must be coordinated with ARRIS engineering—there are reserved values. In the specification process, ARRIS will provide specific documentation of the **ApplicationRequestData** and **ApplicationResponseData** descriptors appropriate for the assigned **oda** property value.

uri is the name of the movie file to be played, such as **titanic.mpi** (required). For a feature movie, this can be found in the CMM metadata **FileInfo::pathname** where **FileInfo::type** is equal to **feature**. For a preview, this can be found in **FileInfo::pathname** where **FileInfo::type** is equal to **preview**.

bitrate is the bit rate in bits/sec as an integer value (6000000 = 6Mbs) (required). For a feature movie, this can be found in the CMM metadata **FileInfo::bitrate** where **FileInfo::type** is equal to **feature**. For a preview, this can be found in **FileInfo::bitrate** where **FileInfo::type** is equal to **preview**.

price is the price of the asset as a string, such as **$2.99**. (optional)

name is the title of the asset. (required)

npt is the number of seconds into the title at which to begin play-out. (optional; defaults to 0)

providerId is the CMM metadata Provider ID. (required)

[redacted]

catId is the CMM metadata **CategoryInfo:modElementId** from the category from which the user purchased the asset. (required)

shouldEncrypt indicates whether or not session-based encryption is to be used, for SA solutions. The value can be either **true** or **false**, and defaults to **false**. This information should come from the CMM metadata **AssetInfo:FileInfo:PropertyCollection:Property:Encryption**. (optional)

collectionId is required by ARRIS to be present in the **ApplicationRequestData** string.

The following is a sample **ApplicationRequestData** string:

```
oda=nable;uri=moviename.mpi;bitrate=6000000;
price=$2.99;name=Good Will Hunting;npt=0;providerId=123;
catId=6452;shouldEncrypt=true;collectionId=1234
```

[redacted]

Use cases

The system goes through several states as it acquires and loads data. As illustrated below, the user agent approves download of either a now/next file, or a big-block file. It never maintains both files in memory—the guide and the SmartBar (now/next) share a single data file, so there is only ever one source for program data.

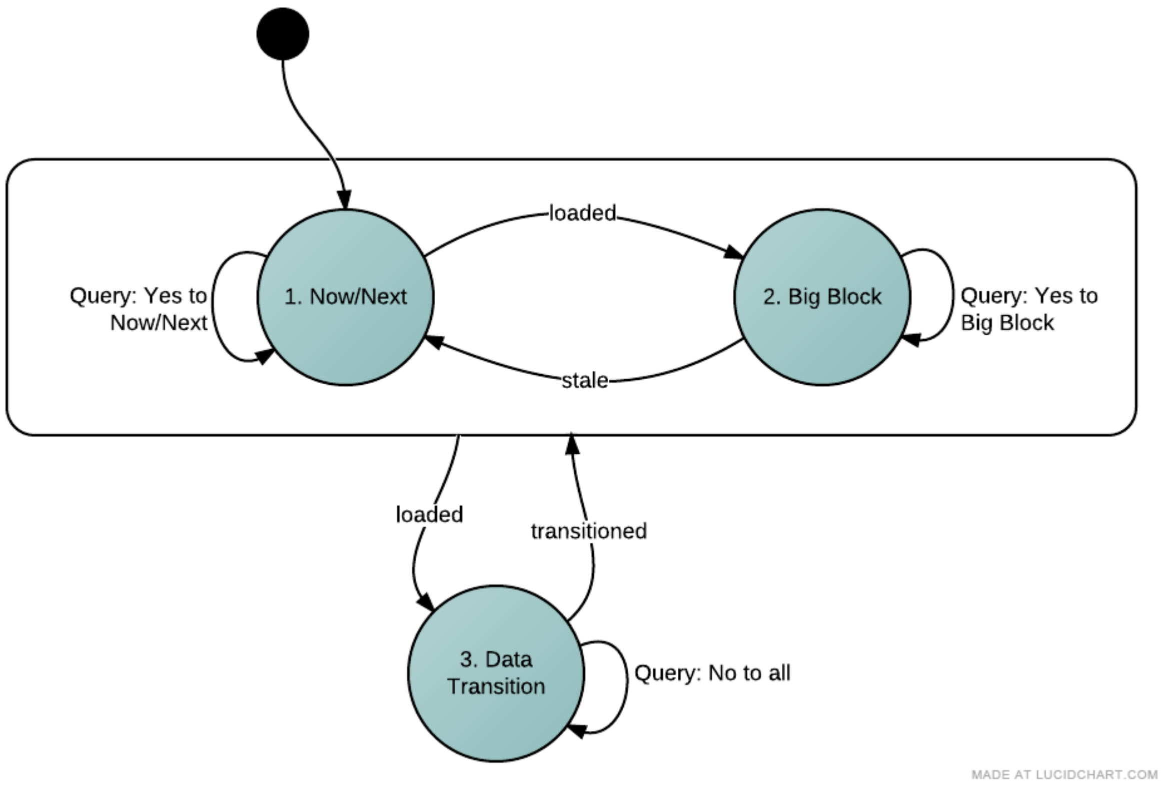
At startup, the user agent waits for the services implementation to find a now/next data file, and requests that it be downloaded. This is the **Now/Next** state.

When the data download is complete, the user agent enters the **Data Transition** state to process the now/next guide data. In this state, the user agent rejects all offers of new data from the services implementation.

When the data is processed and transitioned, the user agent enters the **Big Block** state. In this state, the user agent waits for the services implementation to find a big-block file, and requests that it be downloaded. When the download is complete, the user agent processes the guide data.

At any time, if the guide data (now/next or big-block) becomes stale, the user agent transitions back to the Now/Next state.

These states are illustrated in the following diagram, and further explained in the use cases which follow:



Startup

At startup, when it is in the Now/Next state, the user agent waits for a download notification from the services implementation. When it receives a notification that a current now/next file is

[redacted]

available, it requests the file, and loads the now/next block. Once the now/next data is loaded, the user agent switches into the big-block state in order to load the big (currently 28-hour) block.

Until the guide data is loaded, the guide indicates in each grid row that no data is available.

New big-block data available

When it is in the Big Block state, the user agent receives a download notification from the services implementation, compares names, and recognizes that this is newer than the data it currently has (see “File names” section below). In addition, the data file must cover the current time. The user agent then instructs the services implementation to initiate the download.

If the big block exceeds the device’s storage capacity, the user agent continues to use the existing data. If the big-block data runs out, the user agent enters the Now/Next state, and proceeds as in the Startup use case.

No big-block data available

If the user agent runs out of data while it is in the Big Block state, and there are no current big-block files signaled on the headend (or the currently-signaled file is stale), the user agent enters the Now/Next state, and proceeds as in the Startup use case.

No big-block or now/next data available

If there is no big-block data available, the user agent enters the Now/Next state and proceeds as in the Startup use case. If there is no now/next data available, the user agent simply maintains the Now/Next state indefinitely, and the guide indicates in each grid row that no data is available.

QAM switching

The user agent’s data-file requests are atomic. That is, when the user agent requests a data file, it does not depend on the completion of that request. Due to this, QAM switching does not affect the user agent directly.

If the user agent requests a file, and the viewer tunes to a new channel (which changes the frequency), the vendor’s services implementation discards the data. When the file shows up on the new frequency, the services implementation notifies the user agent of a new download. The user agent requests the file. The services implementation initiates the download, and notifies the user agent when the data is ready for processing. At that point, the user agent loads the data, enters the Data Transition state, and begins using the new guide data.

Encryption

This section covers DTR170.302.u-1 (Encrypt electronic health information). This is steps 8 and 9 on the Self Attestation Form.

FIPS PUB 140-2, Annex A (revision published Jan 4, 2011), *Approved Security Functions for FIPS PUB 140-2*, contains a list of the security functions currently approved by NIST. The approved symmetric-key functions are the Advanced Encryption Standard (AES), the Triple-DES Encryption Algorithm (TDEA), and the Escrowed Encryption Standard (EES).

[redacted] uses 256-bit AES encryption for its data, which also provides FIPS PUB 140-2 approved message-authentication capabilities.

Data encryption and key encoding

[redacted] stores its data in an encrypted XML format. This data is encrypted following the World Wide Web Consortium (W3C) "XML Encryption Syntax and Processing" Recommendation. This Recommendation outlines a process for encrypting all or part of an XML document, or any arbitrary data. [redacted] uses the AES algorithm in the Cipher Block Chaining (CBC) mode with a 256-bit initialization vector, one of the permitted schemes in the Recommendation.

Keys are encoded as base64 strings, as required by the Recommendation.

NOTE: The W3C XML Encryption Recommendation is available at the following location:
http://www.w3.org/TR/xmlenc-core/

Methods and classes

[redacted] uses methods from the following .NET methods and classes to perform data encoding and encryption:

System::Convert.ToBase64String() converts an array of 8-bit unsigned integers to a base64-encoded string. [redacted] uses this method to encode keys. This process is more commonly known as UUencoding.

System.Security.Cryptography.RijndaelManaged provides methods to manage symmetric encryption.

[redacted] are you using Rijndael (which MS documents as deprecated), or are you actually using System.Security.Cryptography.Aes?

System.Security.Cryptography.Xml.EncryptedXml represents the process model for implementing XML encryption. This class implements the W3C XML encryption Recommendation referenced above.

NOTE: For more information on the **ToBase64String()** method, the **RijndaelManaged** class, and the **EncryptedXml** class, please refer to the following Microsoft documents:
http://msdn.microsoft.com/en-us/library/dhx0d524.aspx
http://msdn.microsoft.com/en-us/library/system.security.cryptography.rijndaelmanaged.aspx
http://msdn.microsoft.com/en-us/library/system.security.cryptography.xml.encryptedxml.aspx

[redacted]