Πανεπιστήμιο Θεσσαλίας
Τμήμα Μηχανικών Η/Υ, Τηλεπικοινωνιών & Δικτύων

Διπλωματική Εργασία

Θέμα:

«Ασύρματη μεταφορά αναλογικού σήματος τηλεόρασης από πλατφόρμα windows xp σε pocket pc 2003, με χρήση του UPNP – σε πραγματικό χρόνο»

Επιμελητής:
Ηγούμενος Ιωάννης του Ευθυμίου

Επιβλέπων Καθηγητής:
Κατσαβουνίδης Ιωάννης
(Αναπληρωτής Καθηγητής)

Συνεπιβλέπων Καθηγητής:
Σταμούλης Γεώργιος
(Καθηγητής)

Βόλος,
Μάρτιος 2009

Ευχαριστίες

Ύστερα από μια πορεία πέντε και πλέον ετών στο Τμήμα Μηχανικών Η/Υ, Τηλεπικοινωνιών και Δικτύων του Πανεπιστημίου Θεσσαλίας ολοκληρώνονται οι προπτυχιακές μου σπουδές με την εκπόνηση της παρούσας διπλωματικής εργασίας.

Θα ήθελα, αρχικά, να ευχαριστήσω θερμά τον κ. Κατσαβουνίδη Ιωάννη, Αναπληρωτή Καθηγητή του Τμήματος Μηχανικών Η/Υ, Τηλεπικοινωνιών και Δικτύων, κύριο επιβλέποντα της διπλωματικής εργασίας μου, για τις χρήσιμες συμβουλές και υποδείξεις του καθώς και για την αμέριστη εμπιστοσύνη που επέδειξε στο πρόσωπό μου κατά την εκπόνηση της διπλωματικής μου εργασίας.

Ευχαριστώ επίσης τον συνεπιβλέποντα της εργασίας μου, Καθηγητή του Τμήματος Μηχανικών Η/Υ, Τηλεπικοινωνιών και Δικτύων, κ. Σταμούλη Γεώργιο για την καθοδήγηση του. Καθώς και τον κ. James Kim, μηχανικό της βιομηχανίας στην Κορέα, που στήριξε τη διπλωματική με συμβουλές και εμπειρία.

Από καρδιάς θα ήθελα να ευχαριστήσω τον κ. Δημήτρη Συρίβελη, υποψήφιο Διδάκτορα του Τμήματος Μηχανικών Η/Υ, Τηλεπικοινωνιών και Δικτύων, για την πολύτιμη συνδρομή του στο προγραμματιστικό σκέλος της διπλωματικής εργασίας, όποτε αυτό ήταν απαραίτητο.

Τέλος, ευχαριστώ θερμά την οικογένειά μου για την αμέριστη συμπαράσταση που μου παρείχε όλα αυτά τα χρόνια για την ολοκλήρωση των προπτυχιακών σπουδών μου.

Στις οικογένειες μας

Περιεχόμενα

Περίληψη

Τις τελευταίες δεκαετίες οι εξελίξεις στο χώρο των υπολογιστών και της ικανότητας για μετάδοση πληροφορίας είναι τεράστιες. Οι χρήστες έχουν αντιληφθεί ότι ο προσωπικός τους υπολογιστής αποτελεί μία πολύ δυνατή μηχανή. Απόρροια αυτού είναι η συνεχόμενη απαίτηση για εφαρμογές και εργαλεία που τον διευκολύνουν και τον διασκεδάζουν.

Στην παρούσα διπλωματική εργασία έγινε μια προσπάθεια υλοποίησης ενός λογισμικού που θα μεταδίδει ασύρματα σε ένα τοπικό δίκτυο σήμα αναλογικής τηλεόρασης. Στόχος λήψης, του σήματος, αποτέλεσε η πλατφόρμα των pocket pc, υπολογιστών τσέπης. Όμως οι χρήστες μπορούν να δουν το σήμα και μέσω ενός οποιουδήποτε άλλου υπολογιστή, αρκεί αυτός να είναι εφοδιασμένος με τον windows media player της Microsoft.

Στο πρώτο κομμάτι της διπλωματικής εργασίας αναφέρεται η διαδικασία της υλοποίησης. Το υλικό και το λογισμικό που χρησιμοποιήθηκε. Ο αναγνώστης θα έρθει σε επαφή με τα προβλήματα που υπήρξαν αλλά και με τις λύσεις που δόθηκαν, αν τελικά αυτό ήταν εφικτό.

Στο δεύτερο κομμάτι, στο παράρτημα, βρίσκεται η θεωρία. Αναλύονται βασικές έννοιες που αποτέλεσαν τα εργαλεία της υλοποίησης. Σημαντικότερο τμήμα αποτελεί η αναφορά στο εργαλείο της Microsoft, που ονομάζεται, DirectShow.

Τέλος γίνεται αναφορά στη βιβλιογραφία που χρησιμοποιήθηκε και σε χρήσιμους δικτυακούς τόπους που αποτέλεσαν σημείο αναφοράς.

Καλή διασκέδαση.

ΕΞΟΠΛΙΣΜΟΣ

Για την εκπόνηση της διπλωματικής εργασία χρημοποιήθηκαν τα παρακάτω:

- Φορητός υπολογιστής Macbook 13,3 με επεξεργαστή Intel Pentium Core 2 Duo 2,0 Ghz, 2 Gb RAM – 667. Το λειτουργικό σύστημα που χρησιμοποιήθηκε σαν βάση ήταν Window XP SP3 professional edition.

- Το λογισμικό που χρησιμοποιήθηκε είναι:
    - Visual Studio 2005 with SP1 – C/C++
    - Microsoft Framework 3,5
    - Microsoft Pocket Pc SDK 2003
    - Microsoft Embedded Visual Tools
    - Microsoft Mobile sdk 5.0, 6.0
    - Direct X SDK 2007
    - Windows Media Encoder 9 series
    - Windows Media Encoder 9 sdk
    - Windows Media Format 11 sdk
    - Windows Media Player sdk
    - Windows Media Player for pocket pc sdk
    - Intel UPNP tools

- Pocket pc 2003 HP iPAQ – hx4700
    - window ce 4.21
    - Prossesor - Intel PXA270 (Arm processor)
    - RAM – 64MB
    - Panel : 640 x 480

- Δέκτης τηλεόρασης : Crypto DivaFm (USB)

ΣΤΟΧΟΣ

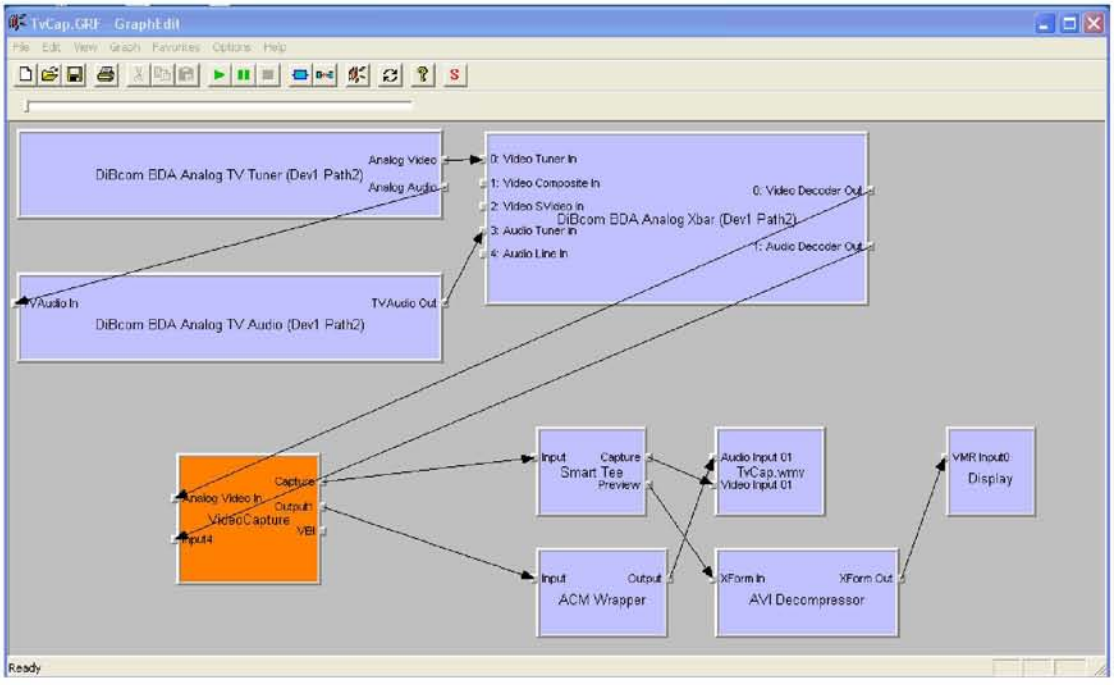
Στόχος της εκπόνησης της διπλωματικής εργασίας αποτελεί η αποστολή σήματος τηλεόρασης από υπολογιστή desktop / notebook σε pocket pc με χρήση του upnp stack, για αποφυγή ρυθμίσεων από το χρήστη. Επιπλέον, δυνατότητα του χρήστη για χειρισμό βασικών λειτουργιών του δέκτη τηλεόρασης μέσω του pocket pc.

1. Signal Capture

1.1Graph Editor

Αρχικά πρέπει να μπορέσουμε να πάρουμε το σήμα που δέχεται ο δέκτης και να το διαχειριστούμε. Τα windows, μέσω του DirectX, και πιο συγκεκριμένα μέσω του DIRECTSHOW μας δίνουν αυτή τη δυνατότητα. Το παραπάνω αποτελεί ένα πολύ δυνατό εργαλείο που βασίζεται στην γενικότερη φιλοσοφία του component programming. Λεπτομέρειες στο παράρτημα.

Χρησιμοποιώντας λοιπόν το εργαλείο αυτό μπορούμε, σε πρώτη φάση, να εντοπίσουμε το δέκτη. Να κάνουμε τις βασικές ρυθμίσεις και να δούμε κάποιο κανάλι. Παρακάτω φαίνεται το γράφημα που προκύπτει στον Graph editor.



Τί σημαίνει όμως το παραπάνω γράφημα?

Τα αντικείμενα αυτά που μοιάζουν με μικρά κουτάκια αποτελούν τα COMponents του συστήματος. Ας τα δούμε ένα προς ένα. Πάνω αριστερά βλέπουμε το αντικείμενο του δέκτη. Μέσω αυτού και του interface που περιέχει μπορούμε να συντονίσουμε το δέκτη και να κάνουμε όλες τις απαραίτητες ρυθμίσεις, . Ακριβώς από κάτω βρίσκεται ο αναλογικός δέκτης ήχου. Αυτός δέχεται ακατέργαστα δεδομένα ήχου από τον αναλογικό δέκτη τηλεόρασης, τα επεξεργάζεται και τα στέλνει στο xbar.

Το xbar θα μπορούσαμε να πούμε ότι αποτελεί την ενδιάμεση πλατφόρμα της συσκευής. Είναι αυτό που διαχειρίζεται τί μπαίνει και τί βγαίνει από αυτή. Οι τρεις αυτές συσκευές αποτελούν το βασικό δέκτη τηλεόρασης. Προγραμματιστικά αυτή που μας ενδιαφέρει περισσότερο στη λήψη του σήματος είναι η συσκευή capture. Αυτή είναι υπεύθυνη για τη λήψη του σήματος που θα μας επιτρέψει στη συνέχεια την επεξεργασία του και την αποστολή του στο δίκτυο.

Πλέον έχουμε το σήμα της τηλεόρασης στα χέρια μας. Μπορούμε λοιπόν να το χειριστούμε ανάλογα και να έχουμε το επιθυμητό αποτέλεσμα.

Παρατηρούμε ότι μετά το αντικείμενο capture, να πούμε ότι τα αντικείμενα αυτά ονομάζονται φίλτρα, υπάρχουν δύο διαδρομές. Η πρώτη επεξεργάζεται το βίντεο και η δεύτερη επεξεργάζεται τον ήχο.

Ακολουθώντας τη διαδρομή του βίντεο βρίσκουμε τα εξής φίλτρα :

- Smart tee : λειτουργεί σαν διακλαδωτής, δίνει το ίδιο σήμα ταυτόχρονα προς όλες τις κατευθύνσεις δίνοντας έμφαση, lossless, στην διακλάδωση που κάνει capture
- avi decompressor : φίλτρο που αναλαμβάνει να αποκωδικοποιήσει το stream τυπου avi (AudioVideoInterlaced)
- VMR : τελευταίου τύπου video renderer των windows, video display
- TvCap.wmv : φίλτρο που αναλαμβάνει το ρόλο της συγχώνευσης εικόνας και ήχου όπως επίσης και των ρόλο της κωδικοποίησης και τελικά την αποθήκευσή του συμπιεσμένου σήματος σε αρχείο. Θα μιλήσουμε αναλυτικά παρακάτω για τη σημασία αυτού του φίλτρου.

Ακολουθώντας τη διαδρομή του ήχου :

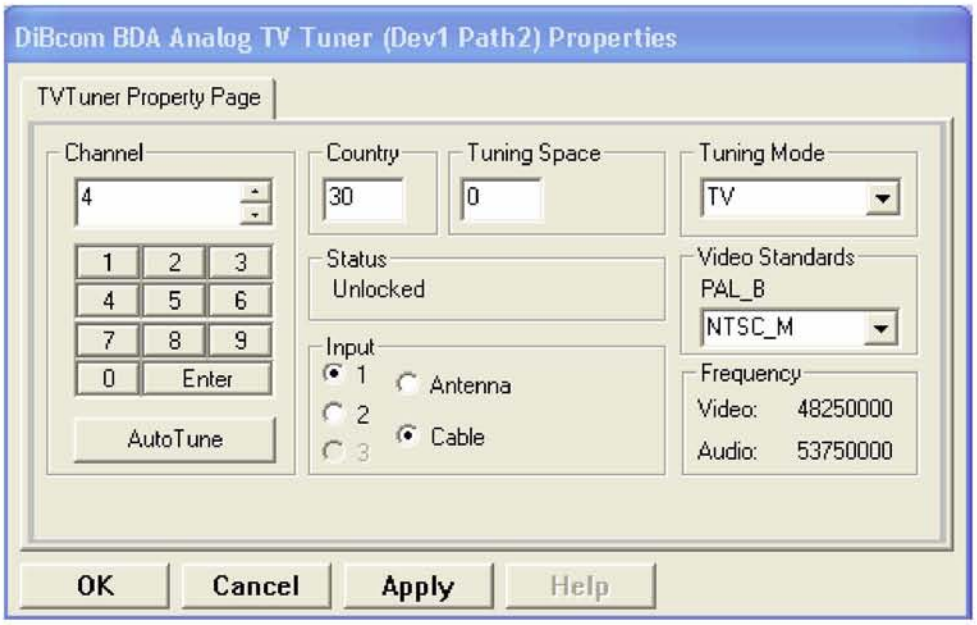
- ACM wrapper : επιτρέπει στα codec που χρησιμοποιούν τον AudioCompressionManager να συμπεριληφθούν στο γράφημα.
- TvCap.wmv : όπως και παραπάνω.

Σε πρώτη φάση έχουμε καταφέρει να δημιουργήσουμε ένα γράφημα που δέχεται σήμα τηλεόρασης, το εμφανίζει στην οθόνη και το αποθηκεύει σε ένα αρχείο. Που πρέπει όμως να επικεντρωθούμε από εδώ και κάτω? Ποιά φίλτρα είναι αυτά που χρειάζεται να διαχειριστούμε για να πετύχουμε το στόχο μας?

Τα φίλτρα που χρειαζόμαστε και θα εξετάσουμε παρακάτω είναι : Analog TvTuner, TvCap.wmv (WMASF writer – windows media advanced system format writer).

1.2 Analog Tv Tuner

Ο αναλογικός δέκτης τηλεόρασης αποτελεί πολύ σημαντικό κομμάτι του γράφου. Είναι το φίλτρο που μας επιτρέπει να έχουμε πρόσβαση στο σήμα τηλεόρασης. Είναι όμως τόσο απλό?



Παραπάνω βλέπουμε αυτό που αποκαλείται property page ενός φίλτρου. Τί μας δείχνει? Στη σελίδα αυτή, που αποτελεί κομμάτι του interface του δέκτη, υπάρχουν όλες οι απαραίτητες ρυθμίσεις που πρέπει να γίνουν ώστε να μπορούμε να πάρουμε εικόνα.

Αρχικά φαίνεται η επιλογή καναλιού. Στο παράρτημα υπάρχει αναφορά για το υπάρχων καθεστώς καναλιών και αντίστοιχων συχνοτήτων στην Ελλάδα.

Πολύ σημαντικό είναι να έχουμε τις σωστές ρυθμίσεις σε όλα τα κουτάκια, δηλαδή στο country βάζουμε 30, στην είσοδο (input) επιλέγουμε cable, στο tuning mode επιλέγουμε tv και στο video standards – ntsc_m.

Η διαδικασία αυτή γίνεται με χρήση κώδικα C++ σε περιβάλλον visual studio.

```
// This call sets the Mode of the tuner in receiving TV
hr = pTvTunerFilter->put_Mode( AMTUNER_MODE_TV );
// This call enables receiving signal from an antenna
hr = pTvTunerFilter->put_InputType( 0, TunerInputAntenna );
// This call sets the appropriate CountryCode, right frequencies
hr = pTvTunerFilter->get_CountryCode(&pCountryCodeGR);
// This call tunes the tuner and stores the channels
hr = TvTuner->StoreAutoTune();
// Tune to ET1(22) - Volos area
hr = pTvTunerFilter->put_Channel( (long)22, -1, -1 );
```

Παραπάνω βλέπετε ένα απόσπασμα κώδικα που κάνει αυτά που περιγράψαμε. Δηλαδή, αφού έχουμε αποκτήσει πρόσβαση στη σελίδα ιδιοτήτων του δέκτη, ορίζουμε τη χώρα και όλες τις άλλες παραμέτρους, βρίσκουμε τα κανάλια, συντονισμός, και επιλέγουμε αυτό που επιθυμούμε.

Κάπου εδώ έχουμε τελειώσει με τις βασικές ρυθμίσεις του δέκτη. Θα χρειαστούμε ξανά το interface αυτού του φιλτρου? Η απάντηση είναι οπωσδήποτε!

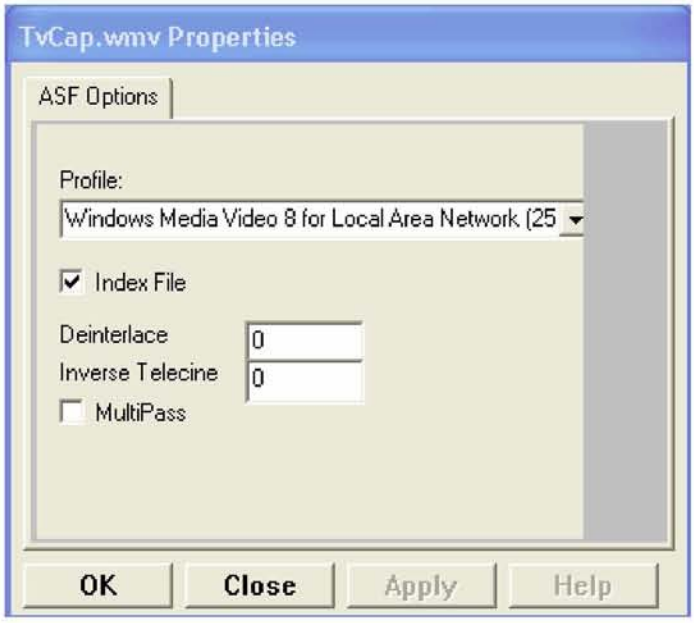
Κάθε φορά που θα χρειάζεται να αλλάξουμε κανάλι ή να αλλάξουμε κάποια παράμετρο θα το κάνουμε μέσω συναρτήσεων που μας παρέχει αυτό.

1.3 WM ASFWriter

Το συγκεκριμένο φίλτρο αποτελεί το πιο σημαντικό στην όλη διαδικασία κωδικοποίησης και αποστολής των δεδομένων. Κομμάτι του media sdk των windows μας δίνει τη δυνατότητα να δώσουμε ως είσοδο δύο ροές, μία βίντεο και μία ήχου, και αυτό αναλαμβάνει την κωδικοποίηση και τη συγχώνευση. Ας πάρουμε όμως τα πράγματα με τη σειρά.

Αρχικά προσθέτουμε το φίλτρο στο γράφο. Αυτό μπορεί να γίνει με χρήση των συναρτήσεων του CaptureGraphBuilder2 interface όπως φαίνεται παρακάτω:

```
// Setting the output stream destination to a filter
hr = pCaptureGraph->SetOutputFileName(&MEDIASUBTYPE_Asf,
                    L"C:\\temp\\tempdir\\TvCap.wmv",
                    &ASFileWriter, &pSinkFilter );
```



Η παραπάνω συνάρτηση παίρνει τέσσερα ορίσματα. Το πρώτο ορίζει των τύπο των δεδομένων που θα χειριστεί το φίλτρο. Το δεύτερο καθορίζει το μονοπάτι στο οποίο θα αποθηκευτεί το αρχείο αλλά και το όνομα αυτού. Το τρίτο και τέταρτο όρισμα αποτελούν δείκτες σε δύο interface, αυτό του φίλτρου εγγραφέα και αυτό των φίλτρων που δέχονται δεδομένα, φίλτρα «νεροχύτες».

Μέχρι στιγμής έχουμε καταφέρει να πάρουμε τα δεδομένα και να τα στείλουμε σε ένα αρχείο κωδικοποιημένα σε μορφή wmv. Πιο συγκεκριμένα, η κωδικοποίηση είναι wmv8 - WMProfile_V80_255VideoPDA το όνομα του GUID. Στόχος είναι να εκπέμψουμε αυτό το σήμα στο ethernet οπότε και οι χρήστες να μπορούν να κάνουν download.

Καλούμε λοιπόν δύο ακόμη αντικείμενα, WriterNetworkSink & WriterPushSink. Το πρώτο αναλαμβάνει να κάνει τις απαραίτητες ρυθμίσεις στο δίκτυο και το δεύτερο αναλαμβάνει την αποστολή των δεδομένων στο δίκτυο.

```
// Set the sink port for the url
DWORD PortNum = 8080;
hr = pWriterNetworkSink->Open( &PortNum );
// Set the host URL, the function returns the url itself, cool haa..
hr = pWriterNetworkSink->GetHostURL(hostURL,&lengthURL);
// Give the network protocol for transmition
hr = pWriterNetworkSink->GetNetworkProtocol(&protocol);
// Set the maximum clients
DWORD MaxClients = 10;
hr = pWriterNetworkSink->GetMaximumClients(&MaxClients);
```

Ας δούμε με τη σειρά των κώδικα που δίνεται. Ορίζουμε την πόρτα επικοινωνίας, δηλαδή την πόρτα μέσα από την οποία θα γίνεται η αποστολή και η παραλλαβή των δεδομένων. Θέτουμε την πόρτα αυτή να είναι η 80 για να μην υπάρχουν προβλήματα με το ανάχωμα ασφαλείας. Στη συνέχεια ζητούμε από το λειτουργικό σύστημα μέσω της εντολής GetHostURL να περάσει την ip του υπολογιστή στης παραμέτρους της εφαρμογής μας, ορίζουμε το πρωτόκολο που θέλουμε να χρησιμοποιήσουμε, HTTP, και το μέγιστο αριθμό πελατών που μπορούν να συνδέονται στο σύστημα.

Το δίκτυο έχει ρυθμιστεί. Το μόνο που θέλουμε πλέον είναι να στείλουμε τα δεδομένα σε αυτό. Τη δουλειά αυτή αναλαμβάνει το δεύτερο από τα δύο αντικείμενα που είδαμε παραπάνω – PushSink.

Πρέπει στο σημείο αυτό να αναφέρουμε δύο έννοιες : PUSH, PULL. Με τη διαδικασία push αναφερόμαστε στα αντικείμενα που σπρώχνουν, κάνουν upload, δεδομένα σε ένα stream. Αυτό μπορεί να είναι από ένα αντικείμενο σε ένα άλλο ή στο δίκτυο. Από την άλλη πλευρά pull σημαίνει παίρνω/ τραβάω δεδομένα από ένα stream.

```
// Connect to the host url
hr = pPushSink->Connect(hostURL, NULL, true);
```

Με την εντολή αυτή λέμε στην εφαρμογή μας να συνδεθεί στη διεύθυνση που ορίσαμε προηγουμένως και να αρχίσει να στέλνει δεδομένα. Τελειώσαμε λοιπόν? ‘Οχι. Γιατί όμως, τί λείπει?
Έχουμε ρυθμίσει το δίκτυο και έχουμε δώσει εντολή στα αντικέιμενα μας να στέλνουν δεδομένα σε αυτό. Πώς θα πάρουν όμως τα δεδομένα?

Τέλος, πρέπει να δημιουργήσουμε στο γράφος μας μονοπάτι που να οδηγεί τα δεδομένα που έχουν γίνει capture από τον αναλογικό δέκτη τηλεόρασης στα αντικείμενα NetworkSink και PushSink που οδηγούν στην αποστολή στο δίκτυο.

Μέσω του interface του asf writer μπορούμε να αποκτήσουμε πρόσβαση σε ένα πολύ σημαντικό εργαλείο που λέγεται IWMWriterAdvanced2. Το συγκεκριμένο interfase επιτρέπει την προσθήκη ή απαλοιφή «νεροχυτών» στο γράφο.

```
// Remove all the sinks
hr = pWriterAdvanced->RemoveSink(NULL);
// Adding sink to the network
hr = pWriterAdvanced->AddSink(pWriterNetworkSink);
// Adding sink to the push sink
// Pushing the data to the server
hr = pWriterAdvanced->AddSink(pPushSink);
```

Η πρώτη γραμμή κώδικα κάνει μία πολύ σημαντική λειτουργία. Αφαιρεί όλες τις διοχετεύσεις από το σύστημα. Δηλαδή?

Διοχέτευση είναι η διαδικασία κατα την οποία δεδομένα στέλνονται σε ένα αντικείμενο που έχει την ιδιότητα του νεροχύτη, sink filter. Αυτό το αντικείμενο συνήθως είναι κάποιο αρχείο ή το δίκτυο. Στην περίπτωσή μας στόχος είναι η αποστολή δεδομένων στο δίκτυο. Είδαμε όμως παραπάνω ότι για να γίνει αυτό πρέπει να δημιουργήσουμε το αρχείο και στη συνέχεια να στείλουμε τα δεδομένα στο δίκτυο. Αν βάλουμε λοιπόν το γράφο να τρέχει θα δημιουργηθεί ένα αρχείο που θα μεγαλώνει συνέχεια σε χωρητικότητα. Μη αποδεκτό, ένα τέτοιο αρχείο θα μπορούσε να οδηγήσει σε κορεσμό της μνήμης. Υπάρχει τρόπος να επιλυθεί το πρόβλημα? Φυσικά. Αφαιρούμε όλες τις διοχετεύσεις και προσθέτουμε αυτές που μας ενδιαφέρουν μία προς μία.

Έχουμε λύση το πρόβλημα του συνεχώς αυξανόμενου αρχείου, πρώτη γραμμή κώδικα, και στη συνέχεια προσθέτουμε τις διοχετεύσεις προς το δίκτυο, δεύτερη και τρίτη γραμμή κώδικα.

Είμαστε, επιτέλους, σε θέση να συνδέσουμε τα κομμάτια του γράφου και να πατήσουμε το play. Όταν το κάνουμε αυτό αρχίζει να εκπέμπετε σήμα στη διεύθυνση που έχουμε ορίσει παραπάνω.

2. TIME DELAY

Ξεκινώντας το γράφο διαπιστώνουμε ότι το σημαντικότερο πρόβλημα είναι η καθυστέρηση. Τί εννοούμε με την έννοια καθυστέρηση? Καθυστέρηση, σε μία διαδικασία broadcast, αποτελεί ο χρόνος που χρειάζεται ο χρήστης για να πάρει τα δεδομένα στην οθόνη από τη στιγμή που πατάμε το κουμπί αποστολής στον αποστολέα.

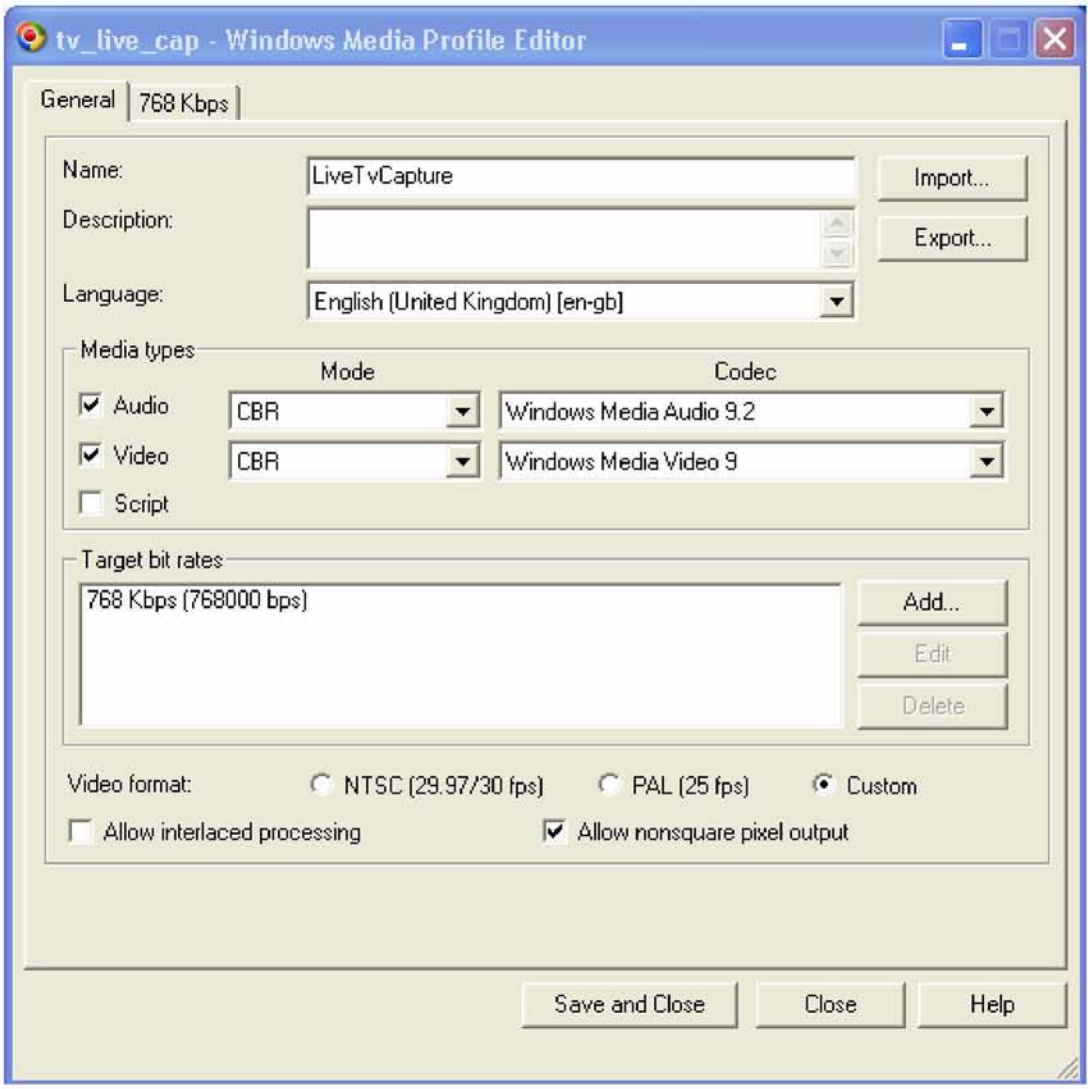
Στο παράρτημα υπάρχει αναλυτική περιγραφεί των χρόνων καθυστέρησης και πως αυτοί προκύπτουν.

Τρεις είναι οι βασικές πηγές καθυστέρησης. Ο κωδικοποιητής, ο εξυπηρετητής και ο player στο κομμάτι του πελάτη.

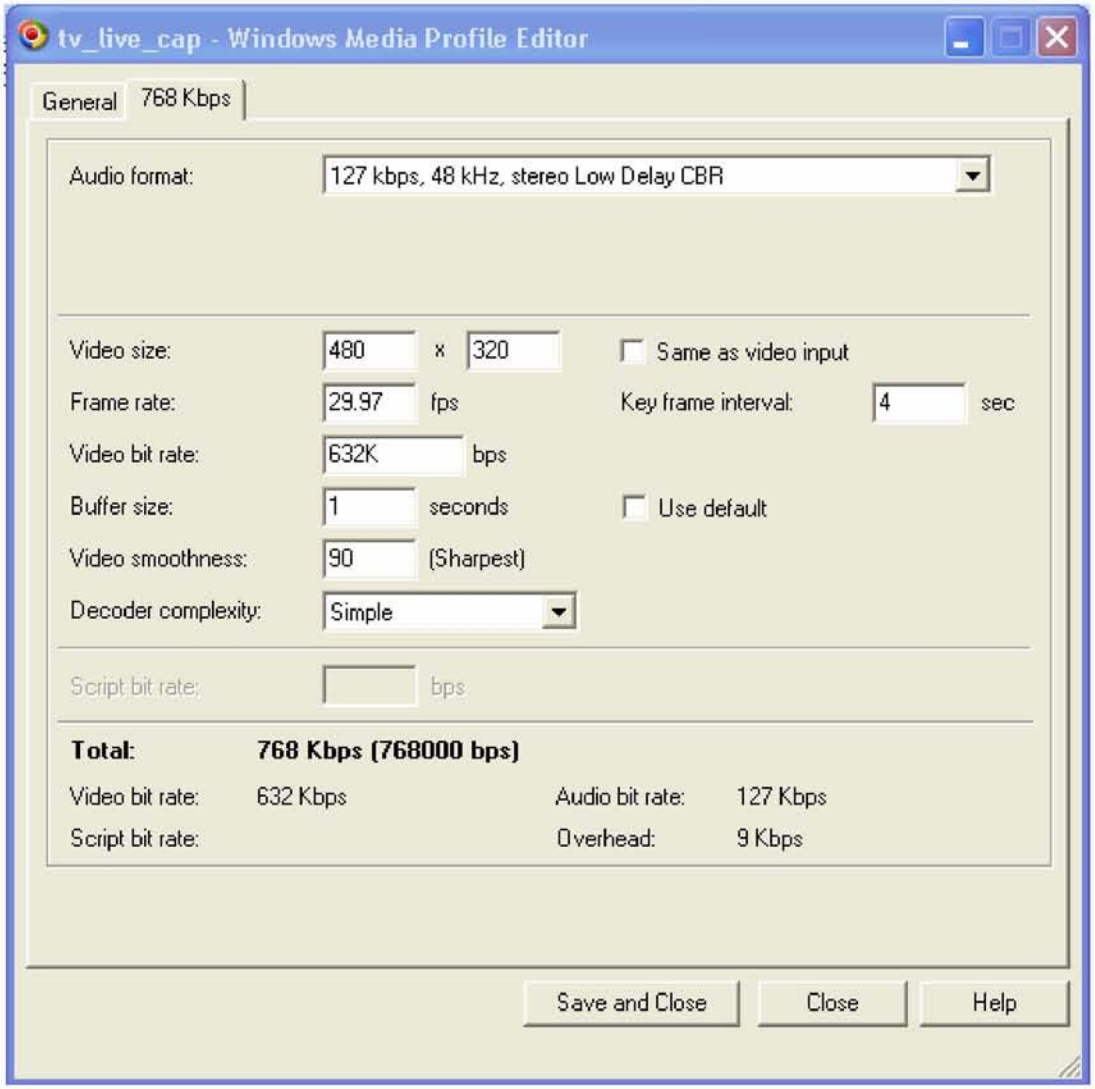
2.1 ENCODER – κωδικοποιητής

Η σημαντικότερη δομή του γράφου. Το αντικείμενο αυτό αναλαμβάνει την κωδικοποίηση των δεδομένων που παίρνει ο γράφος. Το σύστημα, όπως είδαμε παραπάνω ορίζει έναν εναλλακτικό τύπο κωδικοποίησης. Δεν είναι όμως ο επιθυμητός. Το πρωτόκολο που χρησιμοποιείται είναι το wmv8, ενώ υπάρχει διαθέσιμο ήδη το wmv9, το μέγεθος της εικόνας εξόδου είναι μικρότερο από το αναμενόμενο και τέλος και σημαντικότερο εισάγει στο σύστημα ένα buffer delay της τάξεως των 5 δευτερολέπτων.

Η λύση έρχεται από την ίδια τη Microsoft. Από το διαδικτυακό τόπο της ίδιας κατεβάζουμε το πακέτο του wmencoder, windows media encoder. Εκεί περιέχονται οι βιβλιοθήκες για την καινούρια κωδικοποίηση καθώς και ο window media encoder editor. Ένα πολύ χρήσιμο εργαλείο με τη βοήθεια του οποίου δημιουργούμαι το profile που εμείς επιθυμούμε και το φορτώνομαι στο writer. Παρακάτω αναφέρεται η διαδικασία με τη σειρά:



Στην πρώτη σελίδα του encoder editor επιλέγουμε τα βασικά στοιχεία του προφίλ. Αυτό θα διαχειρίζεται βίντεο και ήχο, με σταθερό ρυθμό bit (CBR), και θα το κωδικοποιεί σε windows media audio 9.2 kai windows media video 9. Ο ρυθμός των bit, συνολικό παραγόμενο stream, θα ειναι 768 Kbps.



Στη δεύτερη σελίδα του encoder φαίνονται επιμέρους χαρακτηριστικά.
Στο audio format, κορυφή της σελίδας, υπάρχει η ένδειξη Low Delay CBR. Αυτό σημαίνει ότι ο κωδικοποιητής είναι ρυθμισμένος να κάνει όσο πιο γρήγορα μπορεί. Πολύ σημαντικό στο delay είναι και το μέγεθος του buffer που το έχουμε ορίσει στην ελάχιστη δυνατή τιμή, δηλαδή 1 δευτερόλεπτο.

Οι υπόλοιπες ρυθμίσεις αφορούν αυτό που θα βλέπει ο θεατής. Το μέγεθος της εικόνας, την καθαρότητα και το ρυθμό που εμφανίζονται τα frames.

Δημιουργήσαμε το προφίλ με τα χαρακτηριστικά που επιθυμούμε. Επόμενο βήμα είναι να το « φορτώσουμε » στον asf writer.

```
// Loading the custom profile to the profile pointer for handling
hr = LoadCustomProfile( profile, &pProfile );
```

```
// Retrieving the interfase of the the writer for to configure
hr=ASFileWriter->QueryInterface(IID_IConfigAsfWriter2,
                                                    (void **)&pConfigWriter);

// Adding the custom profile to the writer
hr = pConfigWriter->ConfigureFilterUsingProfile(pProfile);
// Setting one pass encoding to the writer
hr = pConfigWriter->SetParam( AM_CONFIGASFWRITER_PARAM_MULTIPASS, FALSE, 0);
```

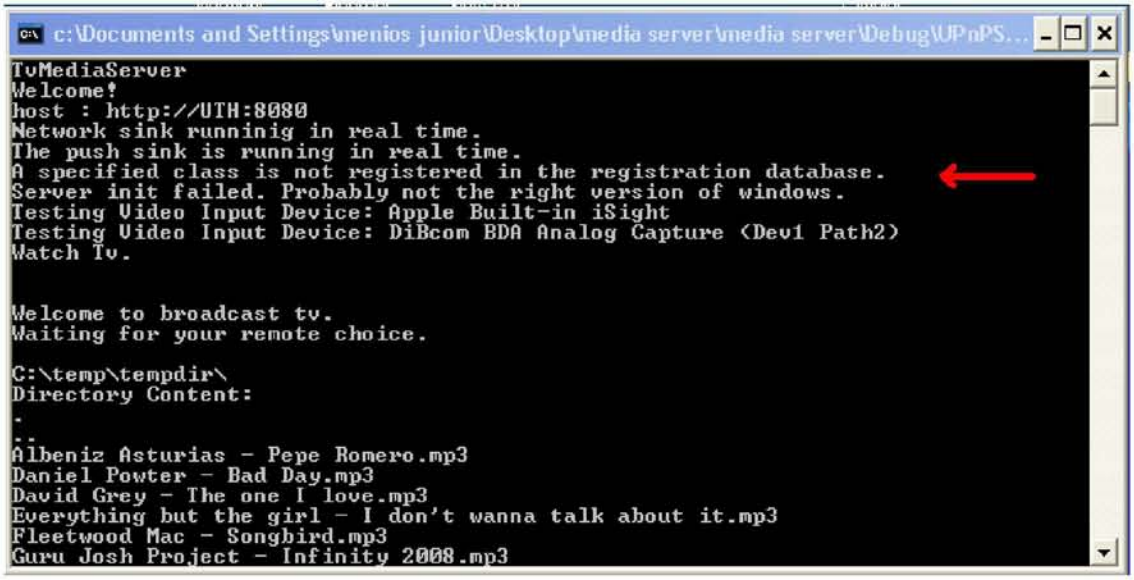
Η εντολή LoadCustomProfile δέχεται ως πρώτη παράμετρο το προφίλ που δημιουργήσαμε νωρίτερα και επιστρέφει ένα δείκτη στο interface του προφίλ αυτού.
Πώς το κάνει αυτό?
Η μεταβλητή profile αντιστοιχεί στο αρχείο .prx που προέκυψε από τον media encoder editor. Η συνάρτηση ανοίγει το αρχείο αυτό και διαβάζει τα στοιχεία του. Το αρχείο είναι σε μορφή XML. Αποθηκεύει όλες τις παραμέτρους που υπάρχουν αποθηκευμένες μέσα στο αρχείο και επιστρέφει, όπως προείπαμε, ένα δείκτη.

Χρησιμοποιούμε το δείκτη για να τροποποιήσουμε το προφίλ του asf writer και να δώσουμε τα δικά μας χαρακτηριστικά. Ορίζουμε επίσης, ότι θέλουμε η κωδικοποίηση να γίνεται σε ένα πέρασμα, 1-pass coding. Αυτή η παράμετρος επιβάλλεται στην περίπτωσή μας, live broadcasting.

2.2 SERVER DELAY

Επόμενο βήμα είναι να απαλείψουμε την καθυστέρηση που δημιουργείται στον εξυπηρετητή. Διαβάζοντας το παράρτημα αντιλαμβάνεται κανείς ότι στην περίπτωσή μας αυτό δεν είναι εφικτό. Το ιδανικό σενάριο είναι να γίνει υλοποίηση της εφαρμογής μας σε πλατφόρμα windows server 2003 / 2008. Αυτές οι πλατφόρμες δίνουν τη δυνατότητα χρήσης του πακέτου window media services 9. Μέσα από τα interface αυτού μπορούμε να ελέγξουμε την καθυστέρηση που δημιουργείτε στο κομμάτι του εξυπηρετητή. Το σύστημά μας βασίζεται στην πλατφόρμα των windows xp sp3 professional.

Τι θα μπορούσαμε να είχαμε κάνει αν όντως είχαμε μία πλατφόρμα τύπου server και γιατί δεν υποστηρίζεται το windows media services 9 sdk, αν και υπαρχει στο Microsoft windows sdk?



Το γεγονός ότι έχουμε κάνει εγκατάσταση το windows sdk δε σημαίνει ότι έχουμε εγκαταστήσει το windows media services. Το γεγονός ότι υπάρχουν τα απαραίτητα αρχεία τύπου .lib και .h σημαίνει ότι έχουν περαστεί στο σύστημά μας οι βιβλιοθήκες που χρειάζονται αλλά δεν έχουν γίνει εγκατάσταση. Ο λόγος είναι, όπως είπαμε και νωρίτερα, ότι έχουμε ένα client maschine και όχι ένα server.

Αν είχαμε όμως να τί θα κάναμε(?):

```
hr = CoCreateInstance(  CLSID_WMSServer,
                        NULL,
                        CLSCTX_ALL,
                        IID_IWMSServer,
                        (void **)&pServer );
// Retrieve a pointer to the IWMSPublishingPoints
// interface and add a new broadcast publishing point.
hr = pServer->get_PublishingPoints(&pPubPoints);
hr = pPubPoints->Add(L"NewPublishingPoint",
WMS_PUBLISHING_POINT_BROADCAST,hostURL, &pPubPoint);
// Query the IWMSBroadcastPublishingPoint interface from
// the newly created publishing point.
hr = pPubPoint->QueryInterface(IID_IWMSBroadcastPublishingPoint,
                               (void **)&pBCPubPoint);

// Set the buffer setting to minimize propagation latency.
hr=pBCPubPoint->
put_BufferSetting(MS_BUFFER_SETTING_MINIMIZE_PROPAGATION_LATENCY);
```

Αρχικά δημιουργούμε το αντικείμενο του server. Μέσω της διεπαφής του αντικειμένου αυτού καλούμε τις υπόλοιπες διεπαφές που υποστηρίζει ο server. Τελικά, και μέσω της διεπαφής IWMSBroadcastPublishingPoint, μπορούμε να

ρυθμίσουμε στο ελάχιστο δυνατό την καθυστέρηση του server. Χρησιμοποιούμε την παράμετρο «MS_BUFFER_SETTING_MINIMIZE_PROPAGATION _LATENCY». Μέσω της παραμέτρου αυτής δίνουμε εντολή στην εξυπηρετητή να ελλαχιστοποιήσει τα δεδομένα που αποθηκεύει, προσωρινά, και τα οποία στη συνέχεια αποστέλλει στον client, έτσι ώστε να μειωθεί ο χρόνος μεταγωγής.

Ας το δοκιμάσουμε. Δημιουργήσαμε ένα σύστημα με windows server 2003, κάναμε εγκατάσταση το απαραίτητο λογισμικό για software development και χρήση του directx. Το δοκιμάζουμε και... τελικά δεν δουλεύει. Τί κάναμε λάθος?

Η απάντηση είναι ότι δεν κάναμε τίποτε λάθος. Το λάθος στην υλοποίηση είναι η πλατφόρμα. Διαπιστώνουμε, ύστερα από μικρή έρευνα, ότι η συγκεκριμένη πλατφόρμα δεν υποστηρίζει πολλά από τα εργαλεία που χρειαζόμαστε, δηλαδή διεπαφές. Το πιο σημαντικό ειναι ο αναλογικός δέκτης τηλεόρασης και η δημιουργία custom profile για τον κωδικοποιητή.

2.3 PLAYER DELAY

Για το κομμάτι του player θα μιλήσουμε παρακάτω. Όσο αφορά όμως το κομμάτι της καθυστέρησης αρκεί μία μόνο συνάρτηση.

```
// Setting buffering time
hr = _AtlModule.spNetwork->put_bufferingTime(buffer_time);
```

Η εναλλακτική που δίνει ο player ειναι 5 δευτερόλεπτα. Με τη χρήση της διεπαφής του ρυθμίζουμε την καθυστέρηση στα 0 δευτερόλεπτα.

Τελικά, και σύμφωνα πάντα και με τον επίσημο δικτυακό τόπο της Microsoft, με τη χρήση wmv είναι πολύ δύσκολο η καθυστέρηση να πέσει κάτω από τα 6 με 8 δευτερόλεπτα. Στην περίπτωσή μας και λόγο της αδυναμίας να ρυθμίσουμε επαρκώς το server τελικά η καθυστέρηση ανέρχεται περίπου στα 15 – 18 δευτερόλεπτα.

Πρέπει να αναφέρουμε ότι στην καθυστέρηση αυτή συμβάλλουν γενικά όλα τα κομμάτια του γράφου και η συνολική διαδικασία του encoding.

3. PLAYER IMPLEMENTATION

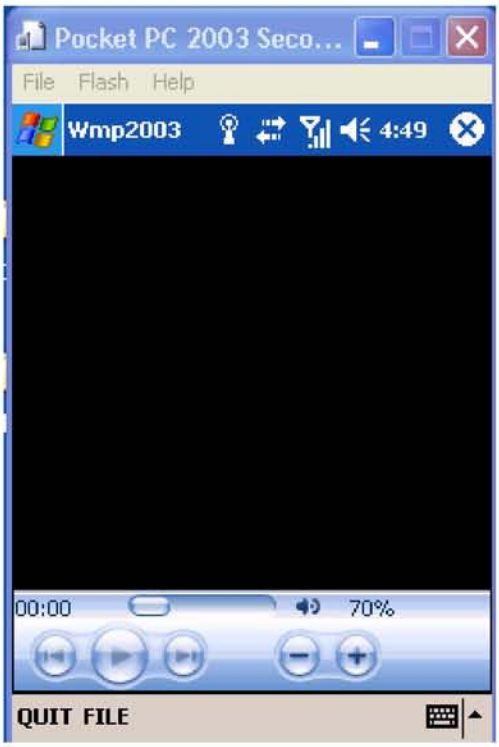
Έχουμε ολοκληρώσει επιτυχώς τη διαδικασία σύλληψης, τη διαδικασία κωδικοποίησης και τελικά τη διαδικασία αποστολής της ροής δεδομένων σε κάποια διεύθυνση στο δίκτυο. Αυτό που χρειαζόμαστε πλέον είναι μία εφαρμογή που θα δέχεται τα δεδομένα και θα τα δείχνει στην οθόνη.

Όπως αναφέραμε και στην αρχή το περιβάλλον του player είναι pocket pc 2003 με windows ce 4,21. Στο περιβάλλον αυτό βρίσκεται ένας πολύ δυνατός player των οποίο θα χρησιμοποιήσουμε, ο window media player 10.

Αυτός ανήκει στη Microsoft η οποία με τη σειρά της δίνει τη δυνατότητα μέσω του windows media player for pocket pc sdk να τον χρησιμοποιήσουμε κατα βούληση στις εφαρμογές μας. Αυτό που πρέπει να κάνουμε είναι να δημιουργήσουμε ένα παράθυρο και να κάνουμε host, δηλαδή να τοποθετήσουμε, τον player μέσα σε αυτό.

Ας δούμε τις ενέργειες με τη σειρά.

```
// Initializing the host window
GetClientRect(hwnd, &rect);
//Create the host window
_AtlModule.WindowDisplay.Create( hwnd,
                                 rect,
                                 TEXT("{6BF52A52-394A-11d3-B153-00C04F79FAA6}"),
                                 WS_CHILD | WS_VISIBLE | WS_BORDER,
                                 0);
```

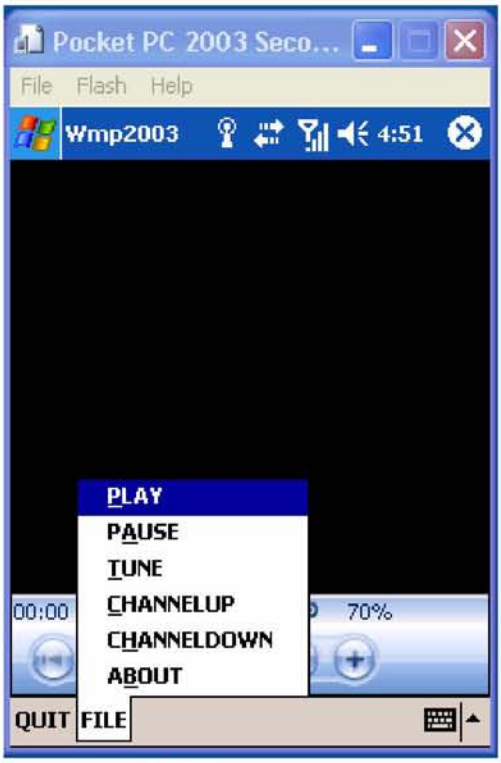


Η πρώτη εντολή μας επιτρέπει να πάρουμε τις συντεταγμένες του παραθύρου που θα αποτελέσει τη βάση, ενώ η δεύτερη μας δημιουργεί το παράθυρο μέσα στο ήδη υπάρχων, parent window. Έχοντας δημιουργήσει τον player μπορούμε να πάρουμε το interface αυτού και να τον διαχειριστούμε.

```
// Retrieve the host window interface that we have just created
hr = _AtlModule.WindowDisplay.QueryHost(&_AtlModule.spHost);
// The media player interface is retrieved
hr = _AtlModule.WindowDisplay.QueryControl(&(_AtlModule.spPlayer));
// Get the network interfase
hr = _AtlModule.spPlayer->get_network(&_AtlModule.spNetwork);
// Get the media player controls
hr = _AtlModule.spPlayer->get_controls(&_AtlModule.spControls);
// Get the media player settings
hr = _AtlModule.spPlayer->get_settings(&_AtlModule.spSettings);

// Set the filepath to the media player
hr = _AtlModule.spPlayer->put_URL( _AtlModule.bstrURL );
```

Τί σημαίνει η κάθε συνάρτηση? Παραπάνω δημιουργήσαμε τον player μέσα σε ένα υπάρχων παράθυρο. Με τις δύο πρώτες εντολές απευθυνόμαστε στο καινούριο αντικείμενο και ζητάμε να πάρουμε τον έλεγχό του. Αυτό γίνεται με την εκχώρηση των συναρτήσεων ελέγχου σε συγκεκριμένους δείκτες. Ο σημαντικότερος δείκτης είναι αυτός που περιέχει όλο το interface του player. Στην περίπτωσή μας ο _AtlModule.spPlayer. Μέσω αυτού, όπως βλέπουμε στις εντολές που ακολουθούν, αποκτούμε πρόσβαση και στις υπόλοιπες διεπαφές του player.



Controls and Settings

Η διεπαφή controls μας δίνει τη δυνατότητα να ξεκινάμε, να σταματάμε και να κάνουμε παύση στα media που βρίσκονται σε κατάσταση αναπαραγωγής. Η διεπαφή setting μας δίνει τη δυνατότητα να ρυθμίσουμε τη φωνή και όχι μόνο ενώ η διεπαφή network μας επιτρέπει να ρυθμίσουμε το μέγεθος του buffer που, όπως αναφέραμε και παραπάνω, επηρεάζει την καθυστέρηση έναρξης παραγωγής των media, όταν αυτά γίνονται download από το δίκτυο.

Πώς όμως θα κάνουμε download από το δίκτυο. Η διεπαφή player μας επιτρέπει να δώσουμε ένα url, διεύθυνση δικτύου, σαν παράμετρο εισαγωγής δεδομένων του player. Αν η διεύθυνση αυτή συμπίπτει με τη διεύθυνση που ο εξυπηρετητής κάνει upload τότε ο player ξεκινάει το download.

Βλέπουμε ότι ο player συνδέεται με τη διεύθυνση και είναι έτοιμος να κατεβάσει και να αναπαράγει τα δεδομένα τις διεύθυνσης αυτής. Πρέπει να αναφέρουμε ότι τα δεδομένα πρέπει να είναι σε μορφή που μπορεί να τα αναγνωρίσει, δηλαδή wmv.

4. UPNP – DLNA CONTRIBUTION

Πριν προχωρήσουμε ας κάνουμε μια μικρή ανακεφαλαίωση. Μέχρι στιγμής έχουμε καταφέρει να πάρουμε σήμα από τον αναλογικό δέκτη τηλεόρασης και να το στείλουμε σε μία διεύθυνση στο δίκτυο. Με τη σειρά του ο player που βρίσκεται στο pocket pc δέχεται σαν παράμετρο τη διεύθυνση αυτή και αρχίζει να κάνει download τα δεδομένα. Χρειαζόμαστε πλέον να απαλλάξουμε το χρήστη από τις δικτυακές ρυθμίσεις της εφαρμογής καθώς και να προσδώσουμε ένα δυναμικό χαρακτήρα στον τρόπο που ο εξυπηρετητής και ο πελάτης προσκολλώνται στο εκάστοτε δίκτυο.

Τη λύση στο πρόβλημα αυτό μας δίνει το upnp stack στην εξελιγμένη του μορφή από τον οργανισμό DLNA. Η intel δημιούργησε μία συγκεκριμένη αρχιτεκτονική μέσω της οποίας συσκευές πάνω στο δίκτυο αναγνωρίζουν η μία την άλλη, χωρίς τη βοήθεια του χρήστη, και αλληλεπιδρούν ανταλλάσσοντας δεδομένα. Στόχος αυτής της αρχιτεκτονικής είναι η εκμετάλλευση δεδομένων media που υπάρχουν σε όλες τις αποθηκευτικές μονάδες ενός οικιακού χώρου. Για παράδειγμα ο χρήστης βρίσκεται στο σαλόνι και έχει συνδέσει την τηλεόρασή του στο δίκτυο. Ταυτόχρονα ένας σκληρός δίσκο βρίσκεται επίσης συνδεδεμένος στο δίκτυο. Ο χρήστης επιλέγει να δει φωτογραφίες ή ταινίες ή να ακούσει μουσική που βρίσκονται μέσα στο σκληρό δίσκο, με το πάτημα ενός κουμπιού και χωρίς να ασχοληθεί με τις ρυθμίσεις του δικτύου.

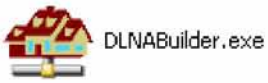
Περισσότερες λεπτομέρειες για την αρχιτεκτονική DLNA υπάρχουν στο παράρτημα.

4.1 SERVER

Στο δικτυακό τόπο της intel κατεβάζουμε το πακέτο Intel Digital Home Device Code Wizard. Μέσα στο πακέτο αυτό αλλά και στο πακέτο Intel Tools υπάρχει ότι χρειαζόμαστε για να προχωρήσουμε στην υλοποίηση μας.

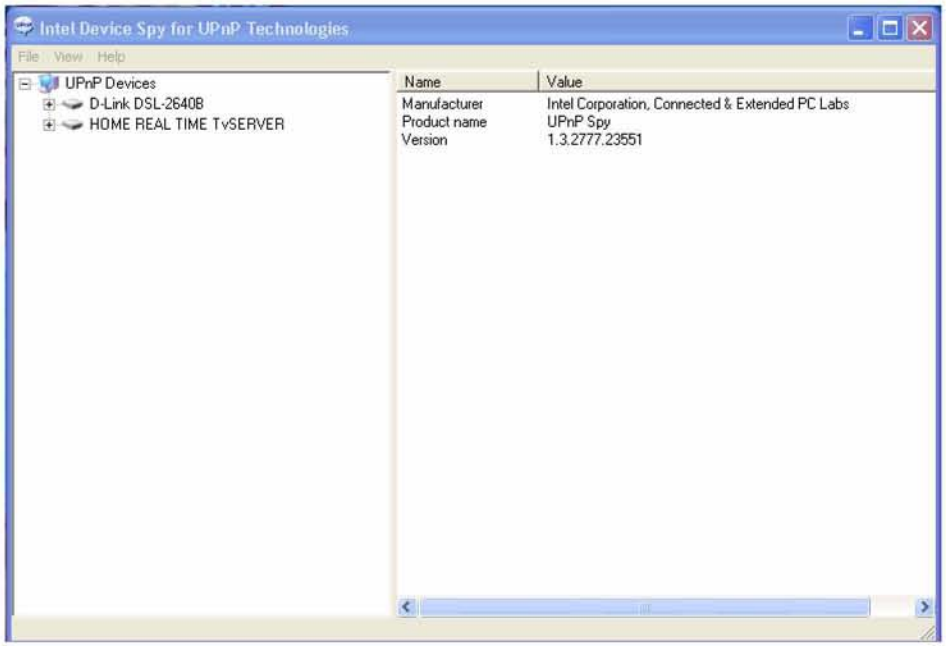
Αρχικά πρέπει να δημιουργήσουμε το template του server. Η αρχιτεκτονική που θα χρησιμοποιήσουμε απαρτίζεται από συγκεκριμένους κανόνες. Για το λόγο αυτό ο βασικός κώδικας παρέχεται έτοιμος από την εταιρία. Εμείς απλά πρέπει να επιλέξουμε την πλατφόρμα που θα χρησιμοποιήσουμε και να κάνουμε τις απαραίτητες διορθώσεις που αφορούν τον μεταγλωτιστή.

Η διαδικασία αυτή γίνεται με τη βοήθεια ενός εργαλείου που λέγεται Device Builder.

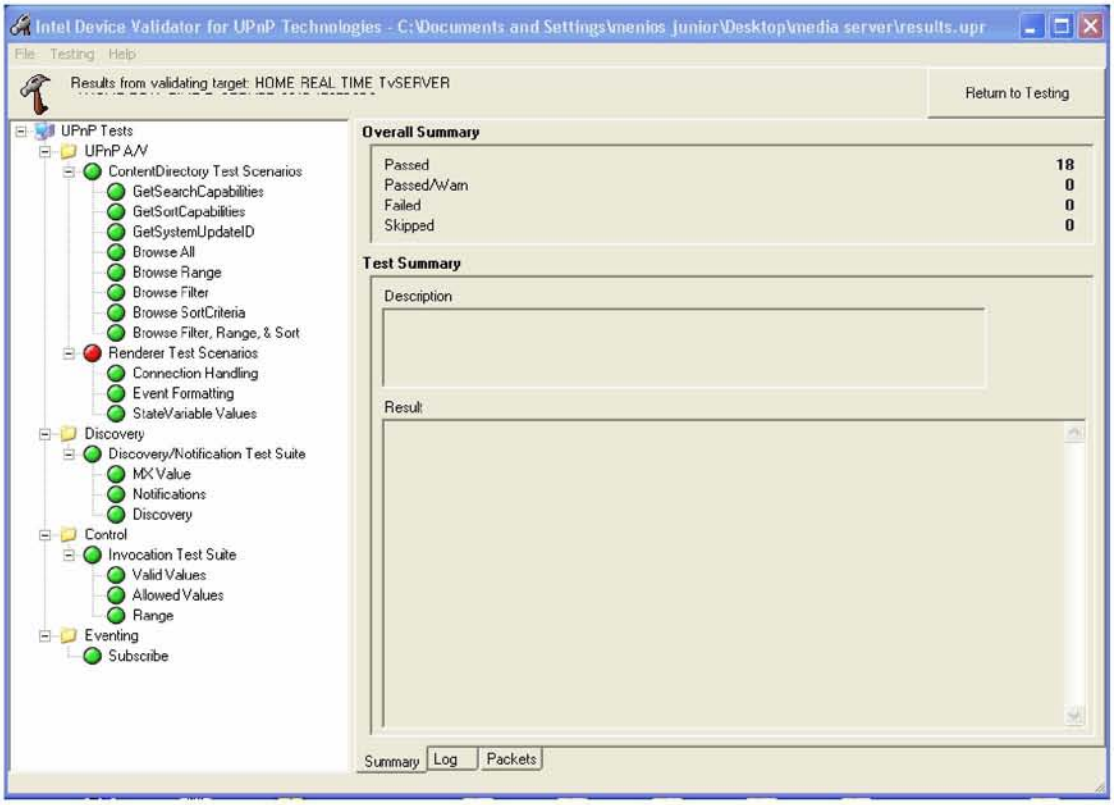


Παραπάνω φαίνεται πώς, με το εργαλείο που προαναφέραμε, επιλέγουμε ότι θέλουμε να δημιουργήσουμε ένα media server. Αν πατήσουμε συνέχεια και ορίσουμε και το κατάλληλο μονοπάτι εξόδου για τον κώδικα, τότε θα έχουμε στα χέρια μας τη βασική υλοποίηση ενός media server σύμφωνα με τις προυποθέσεις του dlna organization.

Μπορούμε πολύ εύκολα να εξακριβώσουμε το παραπάνω χρημοποιώντα δύο ακόμη εργαλεία που κάνουν αυτή τη δουλειά για εμάς. Το Device Spy και το Device Validator, λεπτομέρειες για τα εργαλεία αυτά στο παράρτημα.



Παρατηρούμε ότι το device spy εντοπίζει τη συσκευή. Αυτό σημαίνει ότι είναι επιτυχής η εφαρμογή του upnp stack. Αντιστοίχως στην επόμενη φωτογραφία φαίνεται ότι η συσκευή περνάει όλα τα κριτήρια εγκυρότητας του dlna org.



Καταφέραμε και δημιουργήσαμε, με τη βοήθεια συγκεκριμένων εργαλείων, τη βασική δομή του server που χρειαζόμαστε. Τελειώσαμε? Φυσικά και όχι. Πρέπει να ορίσουμε ένα μοναδικό udn – unique device name, και να ορίσουμε και τον κοινόχρηστο φάκελο, backend init. Αυτές οι ενέργειες αφορούν γενικότερα την υλοποίηση του εξυπηρετητή και όχι συγκεκριμένα την υλοποίηση του tv server.

Παρακάτω φαίνεται το κομμάτι της υλοποίησης που αναφέραμε:

```
if( (backend = dir_search_create()) == NULL){
                printf("Content directory failed.Exiting.\n");
                exit(0);
}
else backendinit.Path = backend;
//ioi: giving the backend diretory path
UdnValue = UdnLabel();
//ioi: returns the UDN of the device (FriendlyName combined with MacAddress)
```

Λεπτομέρειες για τη λειτουργία του κώδικα βρίσκονται στην υλοποίηση.
Τελειώσαμε από το κομμάτι που αφορά τη δημιουργία του server. Υπολείπεται πλέον η προσθήκη του tv signal capture and broadcast που δημιουργήσαμε παραπάνω.

Αν μελετήσουμε την υλοποίηση του DLNA παρατηρούμε ότι βασίζεται σε ένα σύνολο από νήματα που εξυπηρετούν τους πελάτες που έρχονται στο δίκτυο και βιβλιοθήκες από sockets που αναλαμβάνουν τη μεταφορά μηνυμάτων. Εμείς πρέπει αρχικά να δημιουργήσουμε μία διεργασία που θα αφορά την εκτέλεση και το χειρισμό του tv capture. Αυτο γίνεται με τον παρακάτω κώδικα :

```
// Create the graph and run it
hr = ConnectAndPlayGraph();
// Start the controler thread
tv_handle = CreateThread(NULL, 0, &TvServer, NULL, 0, &ptid_tv);
```

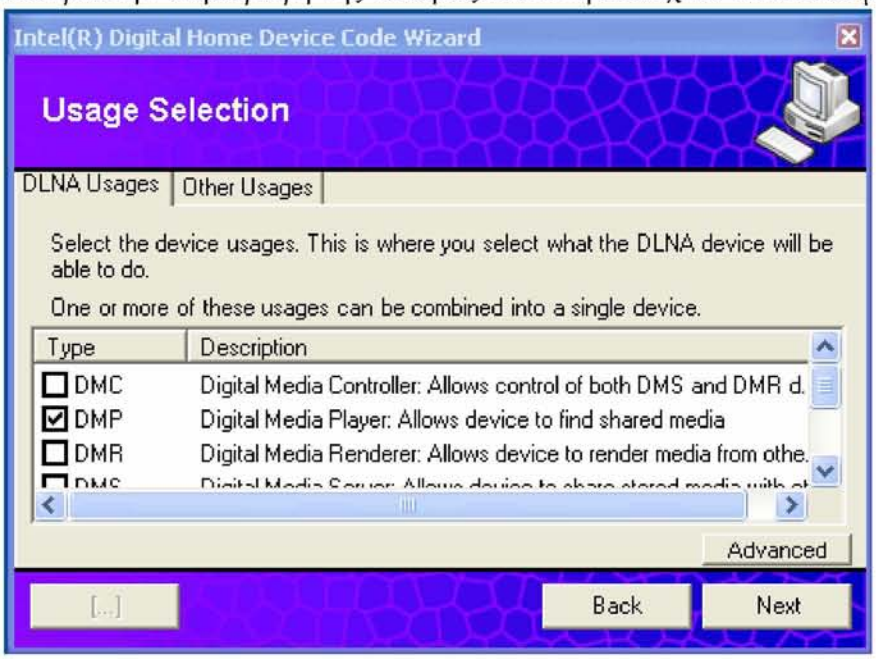
Το κομμάτι του εξυπηρετητή ολοκληρώθηκε. Καταφέραμε συνδέσαμε την εφαρμογή μας με το template της DLNA και ο συνδυασμός αυτός μας δίνει τα εξής:

- Η εφαρμογή μας παίρνει και στη συνέχεια κάνει broadcast σήμα τηλεόρασης σε συγκεκριμένη διεύθυνση URL.
- Ο server μέσα από τις δομές του κάνει broadcast το url

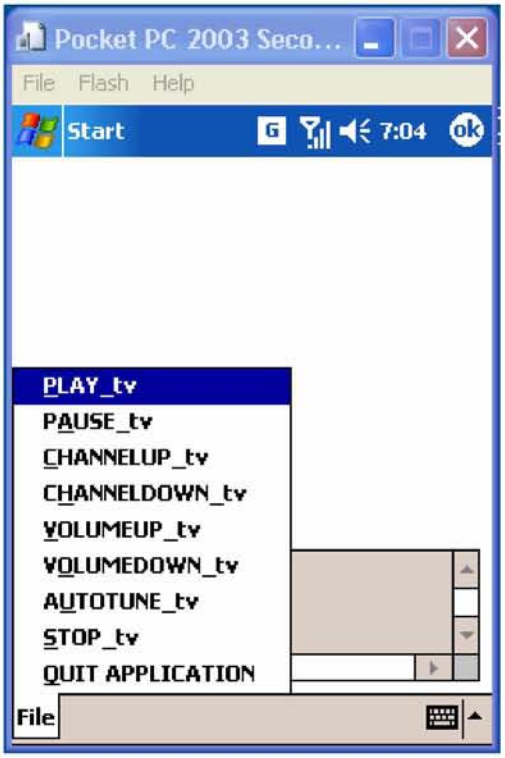
Το μόνο που χρειαζόμαστε πλέον είναι ο κατάλληλος client για να συνδεθεί και να αναπαράγει το σήμα.

4.2 CLIENT

Για την υλοποίηση του πελάτη ακολουθούμε την ίδια διαδικασία. Χρησιμοποιούμε τα εργαλεία της intel για την παραγωγή της στοίβας και στη συνέχεια δουλεύουμε πάνω σε αυτή.



Αφού έχουμε αποκτήσει το template του κώδικα, κάνουμε τις απαραίτητες διορθώσεις που απαιτεί ο μεταγλωτιστής μας. Στη συνέχεια ενσωματώνουμε την έναρξη του player στο κουμπί s PLAY_tv, το κουμπί αυτό είναι ανενεργό όσο ο client δεν είναι συνδεδεμένος στον server. Όπως φαίνεται παρακάτω:



Τα υπόλοιπα πλήκτρα ενεργοποιούνται εφόσον έχει ξεκινήσει ο player και υπάρχει σύνδεση μεταξύ server – client.

Πατώντας λοιπόν ο χρήστης το play, και εφόσον ο client – server έχουν αναγνωρίσει ο ένας τον άλλο, γίνεται έναρξη του wmp και ταυτόχρονα δίνεται σε αυτόν σαν παράμετρος το url από το οποίο πρέπει να κάνει download το σήμα τηλεόρασης που στέλνει ο εξυπηρετητής. Πώς το κάνει όμως αυτό?

Ένα από τα βασικά πρωτόκολλα που χρησιμοποιεί το upnp stack είναι το ssdp, simple service discover protocol. Το πρωτόκολλο αυτό έχει συναρτήσεις για να ανακαλύπτει τις υπηρεσίες που κάνουν broadcast οι εξυπηρετητές. Μαζί με τις υπηρεσίες γίνεται broadcast και το url αυτών. Έχουμε λοιπόν τη δυνατότητα να αποκτούμε κάθε φορά την ip του server μέσω του πρωτοκόλλου αυτού.

Παρακάτω αναφέρεται η συνάρτηση μέσω της οποίας παίρνουμε τη διεύθυνση ip που χρειαζόμαστε.

```
void ILibReadSSDP(struct packetheader *packet,int remoteInterface, unsigned short remotePort,
struct SSDPClientModule *module);
```

όπου το remoteInterface περιέχει τη διεύθυνση του server .

Κάπου εδώ έχουμε τελειώσει. Ο χρήστης μπορεί να δεσμέυσει σήμα τηλεόρασης και να το στείλει στο δίκτυο ενώ ταυτόχρονα, με τη χρήση ενός pocket pc, μπορεί να δει το σήμα αυτό και να εκτελέσει βασικές λειτουργίες του αναλογικού τηλεοπτικού δέκτη.

Καλη διασκέδαση!!!

5. Βιβλιογραφία - Δικτυακοί Τόποι

- Networked Digital Media Standards(A UPnP / DLNA Overview) - Allegro Software Development Corporation
- UPnP AV Architecture: 1 – Document Version 1.00 2
- Universal Plug and Play Machine Models – U. Glasser, Y. Gurevich, M. Veanes
- Programming Microsoft DirectShow for digital video and television - Mark Pesce
- UPNP Design By Example – Michael Jeronimo and Jack Weast
- Introduction to MFC Programming with Visual C++ - Richard M. Jones

- **http://msdn.microsoft.com/el-gr/default.aspx**
    - **libraries**
    - **forums**
- **http://doc.51windows.net/Directx9_SDK/?url=/Directx9_SDK/dir_8.htm**
- **http://intel-software-network.blip.tv/posts?view=archive&nsfw=dc**
- **http://discussms.hosting.lsoft.com/SCRIPTS/WA-MSD.EXE?A2=ind0306c&L=wmtalk&T=0&F=&S=&P=27902**
- **http://software.intel.com/en-us/videos/universal-plug-and-play-tutorial-upnp/**
- **www.intel.com**
- **http://www.upnp.org/**
- **http://www.microsoft.com/communities/newsgroups/en-us/default.aspx?dg=microsoft.public.windowsmedia.sdk&tid=9943dc0e-afe6-4d09-b3da-856bcd452fe8**

Περίληψη Παραρτήματος

Στο παράρτημα που ακολουθεί ο αναγνώστης έχει την ευκαιρία να περιηγηθεί στις βασικές έννοιες που αποτέλεσαν τα εργαλεία για την υλοποίηση της εφαρμογής που παρουσιάστηκε παραπάνω.

Το πρώτο τμήμα του παραρτήματος αφορά την κωδικοποίηση. Αναφέρονται τα ήδη αυτής ενώ ανάμεσα σε αυτά αναφέρεται η κωδικοποίηση που χρησιμοποιήθηκε στην εφαρμογή μας.

Στο δεύτερο τμήμα του παραρτήματος υπάρχει θεωρία που αφορά το βασικό πρόβλημα μίας εφαρμογής πραγματικού χρόνου. Η καθυστέρηση που εισάγεται λόγω εναλλακτικών χώρων αποθήκευσης. Περιγράφονται τα ήδη των χώρων αποθήκευσης αλλά και πώς αντιμετοπίζεται το καθένα για να έχουμε το καλύτερο δυνατό αποτέλεσμα.

Στο τρίτο τμήμα του παραρτήματος γίνεται αναφορά στο εργαλείο του DirectShow. Δίνονται κάποιες γενικές πληροφορίες όπως επίσης και κάποια απλά παραδείγματα. Στόχος είναι ο χρήστης να καταλάβει την έννοια του αντικειμένου αλλά και πως το λειτουργικό της microsoft επεξεργάζεται τα αρχεία των πολυμέσων.

Στο τέταρτο και τελευταίο κομμάτι του παραρτήματος γίνεται αναφορά στη βασική αρχιτεκτονική του upnp – dlna. Περιγράφονται τα βασικά στοιχεία του στοιχεία και η δουλειά στην οποία συμμετέχει το καθένα. Επιπλέον αναφέρονται δύο εργαλεία του upnp που χρησιμοποιήθηκαν για τη δημιουργία της εφαρμογής, upnp device spy και upnp device validator.

ΠΑΡΑΡΤΗΜΑ - ΘΕΩΡΙΑ

A.Microsoft's Windows Media

Microsoft's Windows Media Video has been a leading web video format for many years, but good hands-on information for how to get the best results out of it hasn't always been easy to come by. Also, Windows Media has an enormous breadth of ways it's used, scaling from mobile phones to the PC to CE devices like the Xbox 360. This article strives to codify the most important best practices to get the optimum Windows Media encode, whatever the source and delivery environment.

Update to the Latest Codecs

The first and easiest best practice is to make sure that you have the current versions of the codecs installed on your encoding box. We've recently released a number of backwards-compatible enhancements to the codecs that offer broad improvements for anywhere Windows Media is used. The video codec has seen impressive, fully backwards- compatible improvements. Our codec is now 4-way threaded, meaning it can use all the cores in a dual-core, dual-socket workstation, or one of the new single-socket quad-core systems. Beyond that, there are a variety of general improvements in both performance and quality. More details can be found in "Using Windows Media Registry Keys," pp. 30–32 of the November 2006 issue of Streaming Media. Beyond video, there is also a new version of WMV 9 Advanced Profile, the new WMA 10 Pro, and enhancements to WMA 9, all detailed below.

This article refers to the new and updated versions of WMV 9 as "Version 11" or "v11." There are four ways to get the v11 codecs:

- Install Windows Media Player 11
- Install the Windows Media Format SDK 11
- Install a program that installs the Windows Media Format SDK 11 runtime
- Use Windows Vista (Not approved for use with the Avid DS)

Video Codecs

First, a note on VC-1. There's been some confusion on the relationship between the Windows Media Video codecs and the SMPTE (Society of Motion Picture and Television Engineers) VC-1 spec. VC-1 is the SMPTE designation for their standardization of WMV 9. In essence, you can now think of Windows Media Video 9 as Microsoft's brand for our implementation of VC-1, as implemented for advanced streaming format (ASF) files.

The VC-1 Simple and Main Profiles are progressive, and hence part of the existing WMV 9 profile. The Advanced Profile requires the new WMV 9 AP

implementation. Note that the older and rarely used WMV 9 Complex Profile was not included in the VC-1 spec, and should no longer be used (although files using it are still supported for playback).

Windows Media Video 9

Windows Media Video 9 (WMV 9) has been a popular, mainstream choice for web video applications since the WM 9 ("Corona") launch back in 2002. We've been able to provide several generations of backwards-compatible enhancements to both quality and performance, so today you can encode WMV 9 both faster and with higher quality than ever before.

In general, the only time you'd use a codec other than WMV 9 for web video use is if you were trying to stream native interlaced content (and hence using WMV 9 Advanced Profile, described below) or screen capture content (potentially using WMV 9.2 Screen, also below).

There are two profiles supported in WMV 9—Main Profile and Simple Profile. For computer playback, WMV 9 Main Profile is the optimum choice. Simple Profile is a simpler version of the codec that targets lower-powered mobile devices, like mobile phones. This simplicity means it requires less horsepower to play back, but it also requires more bits to provide equivalent quality to Main Profile since Main has so many more tools it can use to improve compression efficiency. That said, many mobile devices, including those running Windows Mobile 2003 or higher, will decode Main Profile, although they'll need a lower bit rate for it.

Many encoding tools don't provide a profile control, and only do Main Profile.

Windows Media Video 9 Advanced Profile

Windows Media Video 9 Advanced Profile is a non-backwards compatible enhancement to WMV 9, focused primarily on much improved support for encoding content as interlaced, with other enhancements helpful for IPTV and HD DVD. For most web video, neither are needed.

WMV 9 AP was introduced with WMP 10 and the Format SDK 9.5 back in 2004. However, there have been a few format tweaks in the v11 implementation, meaning WMP 9 and 10 will require a codec download to play back v11-encoded WMV 9 AP.

Windows Media Video 9 Screen

The WMV 9 Screen codec is designed for efficient compression of screen recordings. It's a special-use codec—very efficient for this task, but not appropriate for encoding normal video. Screen is the most efficient with simpler, flat graphics, so you can get efficient encoding out of the "Windows Classic" theme in XP and Vista. However, richer graphics environments, especially Vista, include a lot of

gradients and transparencies, and can often encode more efficiently using standard WMV 9.

For efficiency, WMV Screen needs access to the uncompressed RGB source video of the screen shots, and it doesn't work well when compressing from screen shots that have had any lossy encoding already applied to them. It's typically used in conjunction with lossless screen recording products like TechSmith's Camtasia. On a fast computer, it's possible to use Windows Media Encoder to record, or even broadcast, live screen activity with the Screen codec.

Compared to other screen capture codecs, WMV 9 Screen is unique in providing full support for 2-pass VBR and CBR encoding, making it possible to use it for real-time streaming.

Windows Media Video 9.1 Image

The WMV 9.1 Image codec is designed for video sequences made out of individual still images, including transitions. For this special class of content, Image can be quite a bit more efficient than WMV 9, but it is much less efficient for typical motion video sources.

Audio Codecs

Windows Media Audio 9.2

Windows Media Audio (WMA) 9.2 is a fully backwards-compatible upgrade to the venerable WMA standard. It's compatible back to WMP from the mid-'90s (predating ".WMA" files!), and can be played back virtually anywhere. The 9.2 version includes some minor (but welcome) performance and quality enhancements compared to the previous version.

WMA is the general, safe codec choice for any Windows Media file. It's flexible, offers high quality with sufficient bit rate, and is playable by anything that can play a .WMV file. However, there are some scenarios where Voice or WMA 10 Pro can offer better performance.

Windows Media Audio Voice 9

WMA 9 Voice is designed for low bit rate applications below 32Kbps, where WMA and WMA Pro don't perform as well. Despite its name, it does a credible job with music and other non-voice content. You're not going to dance to WMA Voice at low bit rates, but it can intelligibly compress music interludes and sound effects in otherwise voice-centric content.

WMA Voice 9 is supported in WMP 9 and higher. Note that Voice is not currently supported in all non-Windows players. Most notably, it isn't supported in the

current versions of Flip4Mac or the Kinoma player for PalmOS.

WMA Voice replaced the now deprecated ACELP.net audio codec, and should be used instead.

Windows Media Audio 10 Pro

While the new video codec features of WMV are exciting, the biggest technical leap is the new Windows Media Audio 10 Professional codec (WMA Pro 10), which offers up to twice the improvement in compression efficiency compared to WMA 9. The original WMA Pro 9 has been around for several years, and it offers great audio quality and efficiency at 128Kbps and up. WMA Pro 9 supports up to 7.1 channels, up to 24-bit sampling, and up to 96KHz frequencies. But the high minimum bit rate (128Kbps) took it out of the running for most web video tasks. Thus, most streaming video projects have kept using good old WMA 9, which provides good support for lower bit rates.

With WMP 11, we've added a new frequency interpolation mode to WMA Pro, and incremented the name to WMA 10 Pro. With the new mode, we encode a "baseband" version of the audio as normal WMA 9 Pro at half the selected frequency, with additional data that tells how the higher frequencies are added. This gives a stream that's backwards-compatible with the existing decoder, but provides enhanced quality with an updated decoder (bundled with WMP 11).

WMA Pro 10 provides up to two times the efficiency of WMA 9.2, so at 64Kbps it can provide similar quality to WMA at up to 128Kbps. However, if only the old decoder is used, you only get the lower-quality baseband audio.

We feel WMA 10 Pro is the best audio codec included in a major streaming platform today. It handily beats our older codecs, as well as MP3, AAC-LC, and even the new HE AAC-LC, as demonstrated in this study.
WMA 10 Pro is appropriate to use once the majority of your customers are using WMP 11 or another player that supports the full codec. Then at the same bit rate, a WMA 10 Pro stream with the older, non-upgraded decoder can sound a little worse than a WMA 9.2 version. Because it's an enhancement of the older codec, WMA 10 Pro won't trigger a codec download—users only get the new codec if they install WMP 11, Format SDK 11, or if they're running Windows Vista.

The biggest place where WMA 10 Pro is being used today is with Verizon's V CAST mobile media service. We're also working with over a dozen additional vendors to add WMA 10 Pro support to their devices.

Windows Media Audio 9.2 Lossless

The WMA 9.2 Lossless codec is, as the name implies, a lossless audio codec. A

lossless audio codec's output is bit-for-bit identical to its input. Essentially, it's a more efficient alternative to PCM (uncompressed) encoding, and functionally equivalent to Zipping up a .WAV file.

The flip side of lossless encoding is that there's no bit rate control possible—each second of audio takes as many or as few bits as it needs. Hence the codec is only available in Quality VBR mode. Perfect silence takes up very little bandwidth, while white noise takes up as much as uncompressed. Typical savings are around 2:1 for music and 4:1 for TV/movie soundtracks.

In general, WMA 9.2 Lossless shouldn't be used for WMV files for streaming, obviously, as they are CBR. And WMA 9 and 10 Pro can provide incredible sounding audio at lower bit rates, and transparent compression at lower bit rates than WMA Lossless.

Data Rate Modes

Constant Bit Rate (CBR)

In essence, a CBR file is one where the average bit rate (ABR) and peak bit rate (PBR) are identical. This is appropriate for when peak bit rate is the primary constraint, required with real-time streaming, and typically used with devices where the speed of the decoder is the limit.

However, the constant rate is within a certain window of time, the buffer duration. So, a 200Kbps clip with a five-second buffer means that any five seconds of the file has to be at or below 200Kbps per second. But any individual second could go quite a bit higher than that.

1-Pass

1-pass CBR is required for live streaming (webcasting), of course. It's also the fastest bit rate-limited mode available in Windows Media (the bit rate-limited VBR modes are 2- pass only). Note that 1-pass CBR encoding can be significantly improved by the use of the Lookahead registry key parameter, described in a document linked at the end of this article.

2-Pass

2-pass encoding essentially lets the encoder see into the future, ramping the bit rate up and down as needed to account for future changes. With the v11 codecs, it always provides at least as good quality as 1-pass CBR, and will often provide significantly more consistent quality with variable content. And with the v11 codecs, the first pass is much faster than the second pass or a single pass would be, so going to 2-pass doesn't double encode time—it's more typically a 20% increase. So, the 2-pass should generally be used instead of 1-pass for CBR except for live

streaming.

Variable Bit Rate (VBR)

The essential difference between CBR and VBR is that VBR files have a peak bit rate higher than the average bit rate. This lets a VBR file be more efficient for file size, since it can distribute bits throughout the file to provide optimal quality. You can also think of the difference as "CBR maintains bit rate by varying quality, and VBR maintains quality by varying bit rate."

1-Pass Quality-Limited VBR

1-pass quality-limited VBR is a pure VBR—you just specify the quality, and each frame takes as many bits as needed. The final file size can vary tremendously depending on complexity, and there's no limit on peak bit rate at all. Obviously, this makes quality- limited VBR a poor choice for content distribution. However, it's a great mode for archiving content, and since it is 1-pass, it can be captured in real time on a sufficiently powerful box.

2-Pass Bit Rate VBR (Constrained)

2-pass bit rate VBR (constrained) is the optimal encoding mode for when the playback environment can handle a peak bit rate higher than the average bit rate. This is typical of content that is distributed as files instead of streams, be it off a web server or a file on a disc. Almost every web video WMV file not targeted to run on Windows Media Services should be using Constrained VBR.

With previous versions of the codec, we recommended that 1-pass CBR be used for HD content, since there was a problem with occasional dropped frames with 2-pass at high bit rates. This issue has been addressed in the v11 codec, and 2-pass VBR can now be used for HD content. This can result in big savings in file size.

2-Pass Bit Rate VBR

The unconstrained 2-pass bit rate VBR mode is just the same as constrained VBR, but without any peak constraint. This means that playability isn't predictable, and n most cases the constrained mode should be be used instead.

General Parameters

Data Rate

Probably the most critical parameter for any WMV file is its data rate. This determines how big the file is, and how much bandwidth a user would need to stream it. By default, Windows Media measures data rate in kilobits per second (Kbps). To figure out how big a file at a particular data rate is going to be, use this simple equation: the size in KB equals the data rate (in Kbps) divided by 8 times the

duration in seconds. The "8" is in there as data rate is in kilobits per second, but file size is in kilobytes per second. I use "b" for bits and "B" for bytes as shorthand.

Note that Windows Media, like most streaming platforms, uses the correct base-10 metric values for K, M, and B, not the base-two derived values often but erroneously used for shorthand in computing.

To correctly predict the file size, note that you're calculating accuarate KB sizes, while most operating systems list file sizes in KiB/MiB/GiB by default. To get the real value, see the file's Properties and look at the value in bytes

Peak Buffer Size

The buffer-size parameter is available for all bit rate-limited modes. It specifies the duration (in seconds) of the window over which average (for CBR) or peak (for VBR) bit rate is calculated. Bigger buffers can provide more efficient encoding, but make hardware decode more complex and increase latency. For real-time streaming, the maximum startup and random access latency can go up by up to a second for each additional second of buffer.

Peak Bit Rate

Only used in bit rate VBR (peak), the peak bit rate value indicates the maximum bit rate during the peak. Higher values improve the quality of difficult sequences, but also increase the complexity of the decode. A simple way to determine appropriate bit rate is to encode in CBR with the same parameters, and find the highest data rate that will play back reliably. You can then use that as the peak bit rate.

Keyframe Interval

Keyframe interval determines the maximum time between keyframes. A keyframe (also called an I-Frame, for independent or intra frame) is a fully self-contained frame with no other frame based on it. These are the only frames that can be immediately jumped to in random access.

More frequent keyframes make random access, like scrubbing through a file or startup for a streaming file, faster. However, they also reduce efficiency, since a typical keyframe needs more bits to provide equivalent quality compared to normal frames. Note that the codec will insert additional keyframes as needed throughout the file, at points where the video changes dramatically.

Quality

The quality parameter used in the CBR and 1-pass (quality) VBR modes controls the target quality for the video. In quality VBR, it simply controls the quality of each frame, and hence how many bits it gets relative to its difficulty—higher values look better, but take a lot more bits.

In CBR, it controls emphasis on spatial versus temporal quality, essentially setting a minimum bar on frame quality. Lower values let the quality of video frames drop as low as needed in order to hit the target data rate without dropping frames. Higher values set a higher minimum frame quality, and the codec will drop frames on encoding in order to ensure the target bit rate and quality targets are maintained.

Using low quality settings can result in poorer video quality, even when the video is in no danger of dropping frames. This is because the codec doesn't always perfectly adapt to future changes in the video. In general, for most content you want to find the highest value that doesn't drop frames. The v11 codecs, especially if you're using the new registry key settings, can be a lot more efficient than previous iterations, so you can use a higher quality value with the same source and settings than before. I think 75Kbps is a good starting value for most web video now. If you're using Windows Media Encoder, it'll report after the encode if there were any dropped frames, a handy way of seeing if you're using too high a setting. WME also provides reasonable defaults to start with.

Encoder Complexity

Encoder complexity controls how hard the encoder works to encode the video. There are six levels in the v11 codecs with about an eight-fold difference in encoding speed from slowest to fastest, and about a 20% improvement in compression efficiency from fastest to best (which actually isn't the slowest). Different tools present the option in different ways. In Windows Media Encoder, it is set in Tools > Options > Performance. For optimal quality, the second-to-slowest mode (listed as "4" or "80" in some tools, and second last to the left in WME) is almost always the best choice. The slowest mode is a lot slower and very rarely produces any measurable improvement in quality.

Intelligent Streaming

Intelligent Streaming is Microsoft's name for our multiple bit rate (MBR) encoding system. The idea behind MBR is to provide content in multiple bit rates in a single file in order to provide scalability in streaming. Then the player and Windows Media Services (WMS) cooperate to determine the highest data rate version of the video and audio that will fit within the available bandwidth.

Intelligent Streaming is only available when doing real-time streaming from WMS with Windows 2003 Server. Files that won't be coming off WMS should not be encoded with Intelligent Streaming, since it'll just make the files larger, but won't provide any added functionality. Also, not all clients support Intelligent Streaming. It works best when targeting WMP 9 or higher running on Windows.

Tuning for the Target Player

Windows Media Player 11

Windows Media Player 11 is the newest version of WMP. It's available as a free download for Windows XP and built into Windows Vista. With WMP, you can safely use both WMV 9 Advanced profile and WMA 10 Pro's LBR modes.

Windows Media Players 9 and 10

WMP 9 is a good baseline target for mass-market WMV delivery today. WMP 9 is preinstalled with the widely deployed Windows XP Service Pack 2, and it is available via Windows update all the way back to Windows 98. WMP 9 established much of the modern infrastructure for Windows Media, and it's functionally equivalent to WMP 11 for almost all web video. It can safely use WMV 9 and all the WMA 9 audio codecs. WMP 10 was a fine release, but was focused more on performance and media management, and didn't add much functionality specific to web video.

Windows Media Players 6, 7, and XP

Older versions of WMP can use WMV 9 and WMA 9 using codec updates (either automatically downloaded, or via an enterprise deployment). However, none of the pre- WMP 9 versions support the full Intelligent Streaming model implemented in the current version of WMS, and hence can only switch between video tracks of the same frame size, and not at all between audio tracks. WMP 6.4 (the last version for Windows NT 4.0) also isn't reliable with VBR-encoded content.

Flip4Mac

Telestream's Flip4Mac is Microsoft's current recommended solution for Windows Media playback on the Mac.

Flip4Mac handles WMV 9 and WMA 9 and WMA 9 Pro very well, but doesn't currently support WMV 9 Advanced Profile or the other audio codecs. Telestream also offers professional versions of Flip4Mac that support importing WMV into Mac tools like Final Cut Pro and After Effects, and exporting to WMV in QuickTime Pro as well.

WMP 9.1 for Mac

While development has ended on it, WMP 9.1 for Mac is still available. While Flip4Mac provides higher performance and Intel compatibility, WMP 9.1 for Mac supports DRM and some web page embedding options that Flip4Mac does not currently support, although Telestream is working on improving embedding for future versions.

Windows Mobile

Windows Mobile has included Windows Media Player for a number of generations, and any device with Windows Mobile 2003 or later (including 5.0 and the just-released 6.0), has a very capable player, supporting WMV 9, WMA 9, Pro, and Voice. Kinoma Player
Kinoma has recently introduced a new version of the Kinoma Player for PalmOS with support for Windows Media playback. Note that it currently supports only the WMV 9 and WMA 9 codecs, without support for the WMA 9 Voice codec sometimes used for low-bit rate applications.

VLC
The open-source VLC player is adding WMV support to their forthcoming 0.8.6 release. While this product is not supported by Microsoft, it may become a useful solution for WMV playback on other platforms.

WPF/E (Codename)
The Windows Presentation Foundation/Everywhere is Microsoft's codename for a forthcoming rich media browser plug-in. It will support WMV 9 and WMA 9 on Mac and Windows, and within Internet Explorer, Firefox, and Safari. It's currently available as a Community Technology Preview. Downloads and lots of other information are available here.

B.Reducing Broadcast Delay

Introduction
Most computer programs use temporary storage areas in memory called buffers. Typically, a program buffers data before sending it on to a device, such as a hard drive or printer. Microsoft® Windows Media® components use buffers for a number of reasons. Typically, components buffer small chunks of digital media so the data can be processed before rendering it, or sending it to a hard disk drive or computer over a network.

In a broadcast scenario, buffers play a significant role in assuring the quality of digital media as computers and network devices compress, encode, distribute, decompress, and render large amounts of data, very quickly, in real time. In general, the larger the buffer, the better is the end-user experience. However, there is one main disadvantage of large buffers in a live or broadcast-streaming scenario. Buffers cause delays or latencies.

A broadcast delay is the difference in time between the point when live audio and video is encoded and when it is played back, and it is created primarily by the buffers that store digital media data. For example, if the buffers store a total of ten

seconds of data, Windows Media Player will show an event occurring ten seconds late. Often a delay of less than twenty seconds is not a problem. However, when timing is important, Windows Media components provide a number of ways that you can minimize broadcast delay without causing a significant loss of image and sound quality.

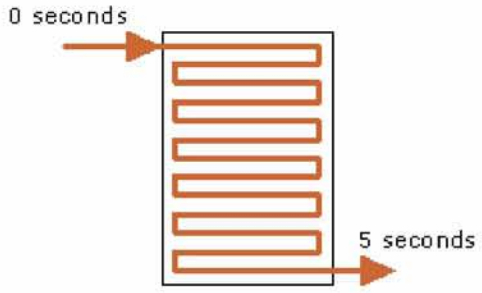
This article describes the buffers and how to adjust them for minimal delay, and the difference between broadcast delay and startup delay. This information is provided in the following topics:

- **Buffering Streaming Media Data**. Describes what buffers are used for and how they work.
- **Minimizing Delay**. Describes how to configure Windows Media components to minimize broadcast delay.
- **For More Information**

Buffering Streaming Media Data

In general when describing the size of a buffer, we usually think in terms of storage space. For example, a word processing program might use 50 kilobytes (KB) to temporarily store a document. When describing a Windows Media buffer, on the other hand, we usually think in terms of time, because audio and video are temporal media. For example, we say a typical buffer holds five seconds of data.

When streaming and playing content, the buffers are constantly changing; new data is continuously being added to the buffer in real time, as older, processed data is sent on and deleted from the buffer. The following figure illustrates how data, in a sense, moves through a five-second buffer.



Data enters the top of the buffer at zero seconds and leaves the bottom of the buffer five seconds later. During the five seconds that the data is in the buffer, a program or algorithm can change, copy, delete, rearrange, or store the data. For example, the encoder buffers data so that the codec can analyze and compress it.

Windows Media stores digital media in buffers for three basic reasons:

- Processing
- Fast Start
- Error correction

Processing

By default, Windows Media Encoder adds a five-second buffer to a session in order to provide the Windows Media Video codec with the optimum amount of data to compress a stream. When playing on-demand content, the buffer is not noticed. However, when encoding a live stream the buffer increases the broadcast delay.

The codec needs a buffer because it analyzes video content over a period of time. For example, it analyzes movement or change from one video frame to the next in order to reduce the bit rate of the content as much as possible and produce high-quality images. To compress data at one point, it might use the analysis of data that occurs two seconds later. Therefore, the more data it can analyze, the better it can compress the stream. Reducing the buffer shortens the delay, but can result in lower quality images.

Fast Start

Fast Start is a new function of the Windows Media 9 Series platform that reduces start-up delay. Start-up delay is different from the type of delay we are talking about in this article. It is the period of time the user must wait for the Player buffer to fill with data when first connecting to a stream. If the buffer is set to hold five seconds of data, the wait is at least that length of time, possibly longer depending on network conditions and the bit rate of the stream.

Fast Start causes data to be sent faster than the actual bit rate of a stream when a user first connects in order to quickly fill the buffer. If the user has high-speed network access, buffering time is reduced and the Player begins playing the stream sooner. After the buffer is filled, the bit rate returns to normal.

In order for Fast Start to work on a broadcast publishing point, which streams data in real time, the server must add broadcast delay to the stream and maintain a buffer. When a user first connects, the buffer is used to send data faster than real time. With Fast Start, the user experience is more like that of watching television, in which he can change channels and instantly view content without start-up delays. The downside is that a few seconds of broadcast delay must be added for the buffer.

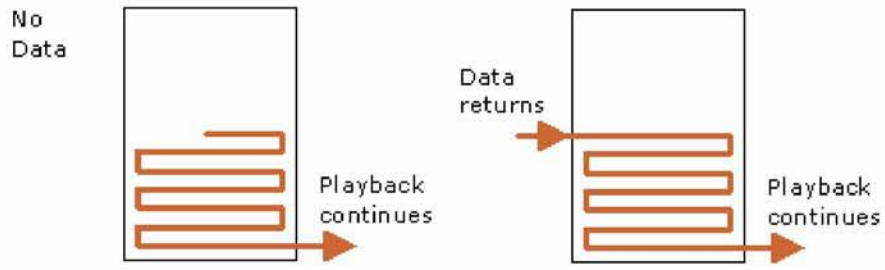
Though the two types of delay are different, in this one sense they are related. In order to reduce start-up delay, Fast Start creates a buffer, which in doing so increases the broadcast delay.

Error Correction

In order for audio and video to play back properly, the digital media must be received in a continuous, unbroken stream. The Internet is an especially challenging network environment. Because of the tremendous size of the Internet, the large amount of data that must be routed and carried, the unpredictable load and bandwidth, and the great distances the data must travel, network conditions are less than favorable for the delivery of streaming content. Though conditions are improving with higher speeds and faster devices, it would not be possible to stream audio and video on the Internet or any congested or slow network without a system of error correction.

Windows Media is designed to deliver the best quality stream possible to users over a wide range of network qualities and connection speeds. To mitigate errors caused by data being lost, delayed, and received incomplete, Windows Media Services and Windows Media Player maintain buffers. Streaming media data is sent in numbered packets that include the addresses of the sender and receiver. The Player uses a buffer to arrange the packets that arrive according to their numbers. If a packet is broken or does not arrive in time, the Player requests a new packet from the Windows Media server. The server can then resend the packet from its buffer.

The following figure illustrates that even when incoming data stops, playback can continue because it uses the data left in the buffer. When incoming data continues it is added to the buffer and presentation continues smoothly even though packets might have been received in the wrong order and at different times.

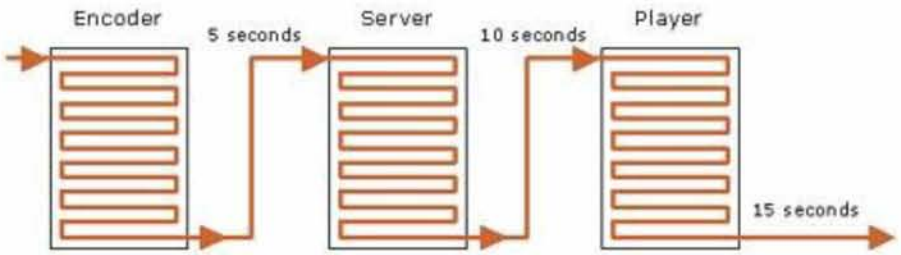


If packets do not arrive in time, the Player edits out the missing data from the buffer. There is a small jump in the presentation where the missing data is, but without the buffer there would be a hole in the presentation.

Minimizing Delay

Buffers are an essential part of creating, delivering, and rendering streaming media content, enabling Windows Media components to provide a high-quality end-user experience. However, there are trade-offs. The larger the buffer size, the longer the broadcast delay. The default buffer time values in Windows Media components

balance quality and delay for most situations. However, there are three buffer settings you can modify to minimize broadcast delay. The following figure shows the buffers in the encoder, server, and player computers, and how each adds to the delay.



The encoder buffers content to enable the codec to optimize compression; the server buffers content to enable Fast Start; and the Player buffer provides smooth playback for the user. Assuming each buffer stores five seconds of data, and taking into account delays that might be added by network devices, the total broadcast delay can be between 15 and 20 seconds. You cannot eliminate the buffers completely; Windows Media components could not work without any buffers. However, by minimizing the buffer sizes using the procedures in the next three sections, you can reduce broadcast delay to approximately six to nine seconds, depending on network conditions.

As you reconfigure settings, test playback and watch for problems. For example, when you reduce the encoder buffer, make sure you are comfortable with the quality of the images. The codec has less motion data to work with, so you might notice more video artifacts, and movement might not appear as smooth. If your content does not include fast motion scenes, you may not even notice any degradation of quality. With a small Player buffer size, playback is more likely to be interrupted by buffering, which is caused by the Player running out of data. Buffering might not be a problem for users with stable, high-speed Internet access. However, users with slow connections may actually need to increase their buffer size. Remember you are balancing quality with broadcast delay, and by reducing delay you may have to make compromises. If you do not have to minimize delay, use the buffers.

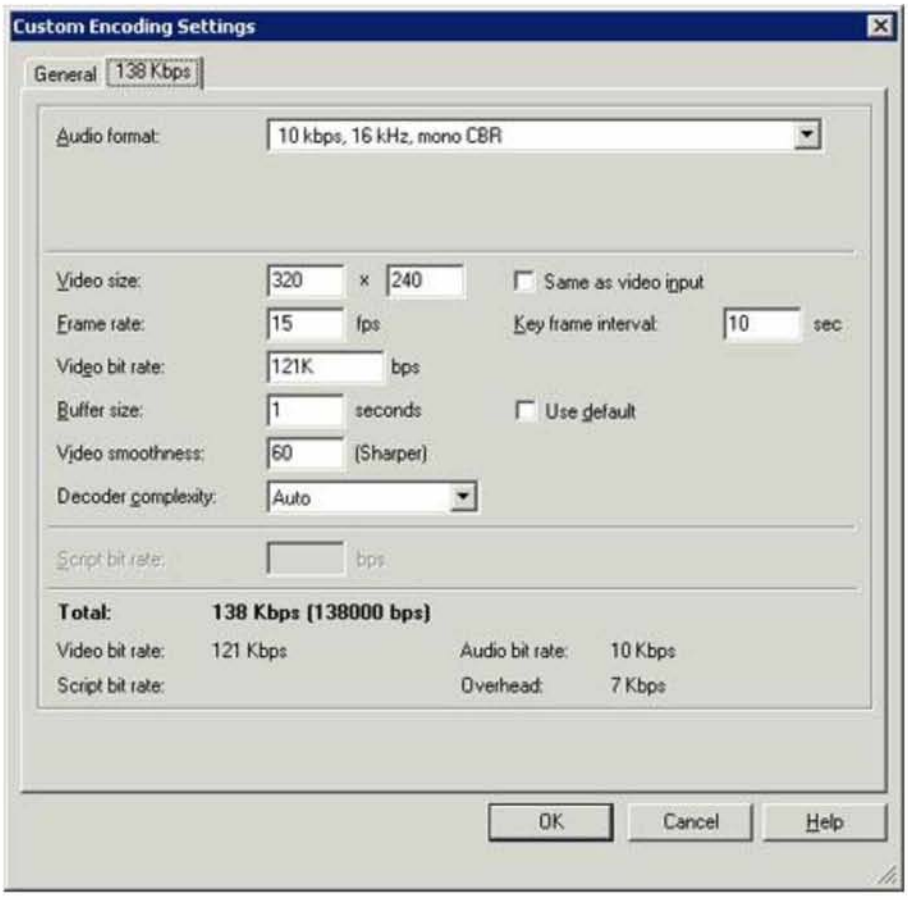
To reduce overall broadcast delay, you can reduce the size of the encoder and Player buffers, and disable the Fast Start buffer on your Windows Media server.

Reducing the Encoder Buffer

Start Windows Media Encoder 9 Series and create or open a broadcast session with **Windows Media server (streaming)** as the destination. In this procedure, we will

assume **Multiple bit rates (CBR)** is selected for both **Audio** and **Video**. However, other settings can be used. Do the following:

1. In **Session Properties** on the **Compression** tab, click **Edit**. **Custom Encoding Settings** opens.
2. Click the **bit rate** tab.
3. In **Buffer size**, type **1**, which is the minimum buffer size. The following figure shows the setting on the **bit rate** tab.For applications needing audio, the end-to-end latency can be reduced by using one of the encoder "low delay" encoding modes available in versions 9.1 of Windows Media Audio and Windows Media Audio Professional and later. These low delay modes cut down on the audio buffering needed both at the encoder and the decoder (player), thereby improving the end-to-end latency. These modes are available when Windows Media Format 9.5 SDK or later is installed on the encoding machine. The WMA/WMA Pro bitstreams encoded with this configuration (WME9 + FSDK9.5 or later) are fully compliant with older WM players, and no codec download by the players is needed. Once FSDK9.5 or later is installed, in the **Compression** tab of WME9, hit the **Edit** and select for audio the "CBR" mode and Windows Media Audio 9.1 codec. In the **bitrate** tab, in the drop-down list for **Audio Format**, select one of the few selections marked as "Low Delay CBR".
4. The Low Delay CBR modes enabled in "Windows Media Audio 9.1 Professional or later" could also be used. Note that the Windows Media Format SDK 11, which is installed when you install Windows Media Player 11 Beta, will also upgrade the WMA Professional codec to WMA 10 Professional.

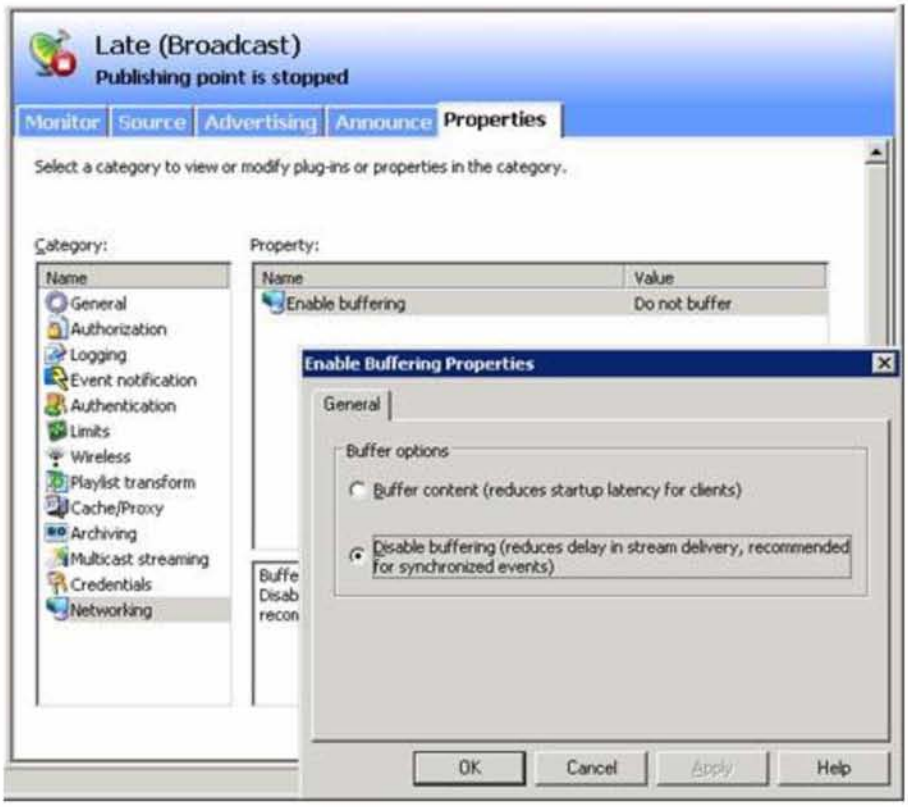


Note Codecs can require different minimum buffer sizes. Therefore, the broadcast delay in most cases is longer than one second. For example, the Windows Media 9 Audio Voice codec introduces a longer delay than the standard Windows Media Audio 9 codec. If you do not require audio, you can further minimize delay by setting Audio Format to 0 kbps, 8 kHz, mono CBR.

Disabling the Server Buffer and Fast Start

Open the Windows Media Services snap-in for Microsoft Management Console (MMC) or Windows Media Services Administrator for the Web, and connect to your Windows Media 9 Series server. Open or create a new broadcast publishing point. Then do the following:

1. Click the broadcast publishing point, and then in the details pane on the right, click the **Properties** tab.
2. In **Category**, click **General**.
3. In **Property**, click **Enable Fast Cache**, and then click the **Disable** button.
4. In **Category**, click **Networking**.
5. In **Property**, click **Enable buffering** and then click the **Properties** button.
6. On **Enable Buffering Properties**, click **Disable buffering**. The following figure shows the Windows Media Services snap-in with **Enable Buffering Properties** open and buffering disabled.

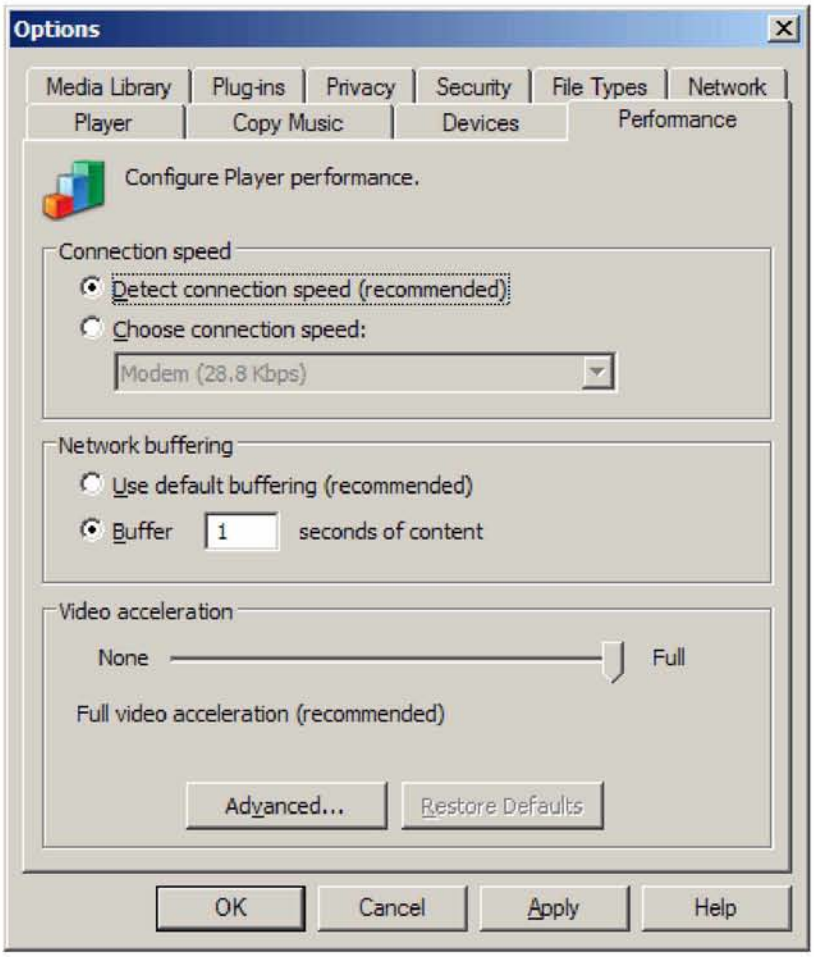


Note Fast Start is disabled at the publishing point level, so keep the Fast Start buffer on publishing points that do not require minimum broadcast delay.

Reducing the Player Buffer

Open Windows Media Player 9 Series, and do the following:

1. On the **Tools** menu, click **Options**.
2. On the **Performance** tab, click **Buffer**, and type **1**, which is the minimum buffer size value. The following figure shows the **Performance** tab with the new setting.



C. DIRECTSHOW

In the early 1990s, after the release of Windows 3.1, a number of hardware devices were introduced to exploit the features of its graphical user interface. Among these were inexpensive digital cameras (known as webcams today), which used charge-coupled device (CCD) technology to create low-resolution black-and-white video. These devices often connected to the host computer through the computer's parallel port (normally reserved for the printer) with software drivers that could handle data transfer from the camera to the computer. As these devices became more common, Microsoft introduced Video for Windows (VFW), a set of software application program interfaces (APIs) that provided basic video and audio capture services that could be used in conjunction with these new devices. Although VFW proved to be sufficient for many software developers, it has a number of limitations; in particular, it's difficult to support the popular MPEG standard for video. It would take a complete rewrite of Video for Windows to do that.

As Windows 95 was nearing release, Microsoft started a project known as "Quartz," chartered to create a new set of APIs that could provide all of Video for Window's functionality with MPEG support in a 32-bit environment. That seemed straightforward enough, but the engineers working on Quartz realized that a much broader set of devices just coming to market, such as digital camcorders and PC-based TV tuners, would require a more comprehensive level of support than anything they'd planned to offer in a next-generation tool. Everywhere they looked, the designers of Quartz realized they couldn't possible imagine every scenario or even try to get it all into a single API.

Instead, the designers of Quartz chose a "framework" architecture, where the components can be "snapped" together, much like LEGO bricks. To simplify the architecture of a complex multimedia application, Quartz would provide a basic set of building components—known as filters—to perform essential functions such as reading data from a file, playing it to the speaker, rendering it to the screen, and so on.
Using the newly developed Microsoft Component Object Model (COM), Quartz tied these filters together into filter graphs, which orchestrated a flow of bits—a stream—from capture through any intermediate processing to its eventual output to the display. Through COM, each filter would be able to inquire about the capabilities of other filters as they were connected together into a filter graph. And because Quartz filters would be self-contained COM objects, they could be created by third-party developers for their own hardware designs or software needs. In this way, Quartz would be endlessly extensible; if you needed some feature that Quartz didn't have, you could always write your own filter.

The developers of Quartz raided a Microsoft research project known as "Clockwork," which provided a basic framework of modular, semi-independent components working together on a stream of data. From this beginning, Quartz evolved into a complete API for video and audio processing, which Microsoft released in 1995 as ActiveMovie, shipping it as a component in the DirectX Media SDK. In 1996, Microsoft renamed ActiveMovie to DirectShow (to indicate its relationship with DirectX), a name it retains to this day.

In 1998, a subsequent release of DirectShow added support for DVDs and analog television applications, both of which had become commonplace. Finally, in 2000, DirectShow was fully integrated with DirectX, shipping as part of the release of DirectX 8. This integration means that every Windows computer with DirectX installed (and that's most PCs nowadays) has the complete suite of DirectShow services and is fully compatible with any DirectShow application. DirectX 8 also added support for Windows Media, a set of streaming technologies designed for high-quality audio and video delivered over low-bandwidth connections, and the DirectShow Editing Services, a complete API for video editing.

Introducing GraphEdit

The basic elements of DirectShow applications—filters, connections, and filter graphs—can be easily represented visually, and drawing a diagram of a DirectShow filter graph can be an important aid in the design process. GraphEdit can be thought of as a whiteboard on which prototype DirectShow filter graphs can be sketched. However, because GraphEdit is built using DirectShow components, these whiteboard designs are fully functional, executable DirectShow programs. More than just a design tool, GraphEdit is a rapid prototyping environment for DirectShow. Any DirectShow application, regardless of complexity, can be built and tested in GraphEdit before you write a single line of application code.

To launch GraphEdit, select Programs from the Start menu, select Microsoft DirectX 9.0 SDK, select DirectX Utilities, and then select GraphEdit. When the program launches, you'll be presented with a large blank area, representative of an empty filter graph with no component filters, as shown in Figure 2-1.

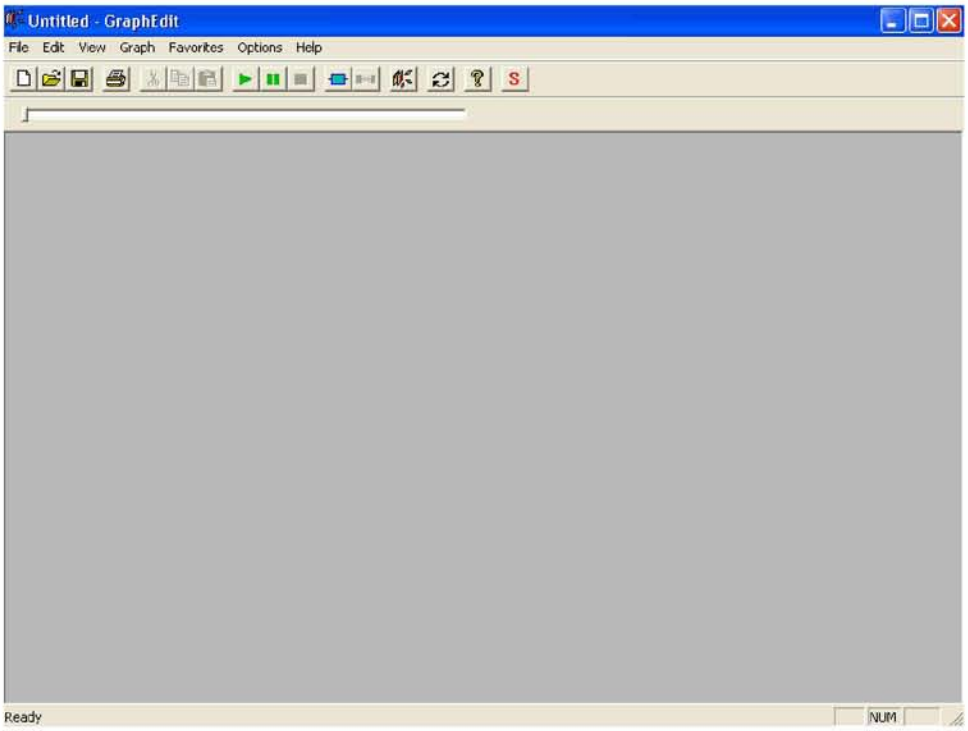


Figure 2-1. GraphEdit's startup state

Rendering Media Files

GraphEdit makes it easy to render a wide range of media files. From the File menu, select Render Media File. You'll be presented with an Open File dialog box asking you to select a file to be rendered. Select the file John Boy 9 – Foggy Day.wav (included on the CD-ROM). If your DirectX installation is correct, you should see the the filter graph shown in Figure 2-2.

GraphEdit has created a complete filter graph, with three components. From left to right, the filter graphs are

- A source filter that points to the WAV file
- A transform filter that parses the file into a series of samples
- A renderer filter that passes the stream along to the default DirectSound device

As explained in Chapter 1, these three types of filters—source, transform, and renderer—can be found in nearly every filter graph. The filter graph created by GraphEdit to render the WAV file is minimal but complete. To hear the WAV file being rendered to the speakers (or whichever DirectSound device has been selected as the default), click the Play button on the toolbar, immediately below the menu bar, as shown in Figure 2-3.

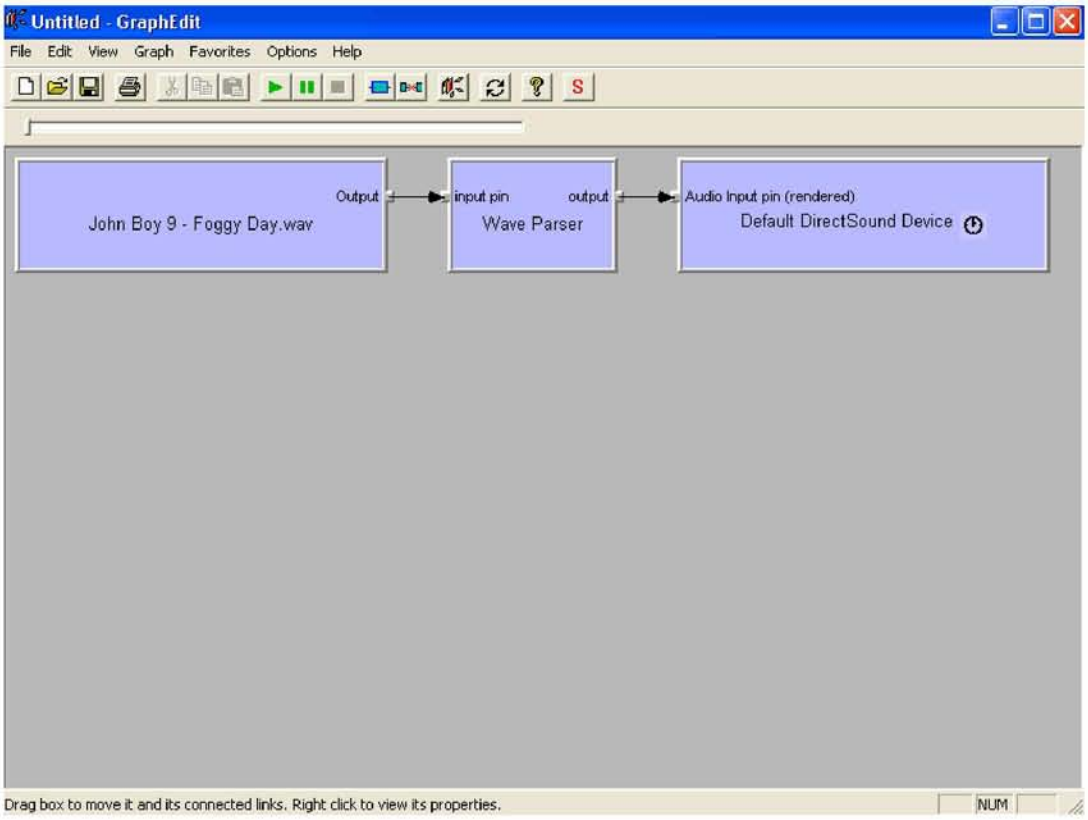


Figure 2-2. Filter graph to render a WAV file

When you click the Play button, the WAV file begins to render and the control below the toolbar begins to fill with color, corresponding to how much of the WAV file has been read into the filter graph. You can pause the filter graph by clicking Pause or stop it by clicking Stop. (As you might expect, the Play, Pause, and Stop buttons send the run, pause, and stop messages to the filter graph.) After the entire stream has rendered to the DirectSound device, GraphEdit stops the filter graph.

Using the Render Media File option, a broad range of media files—all the types understood by DirectShow—can be rendered. In fact, trying to render these files is a good test of the DirectShow capabilities of your system. If you open a media file and DirectShow can't render it, it's likely that you don't have the appropriate collection of DirectShow filters (which might encourage you to write a DirectShow filter that could render the media type you're interested in).

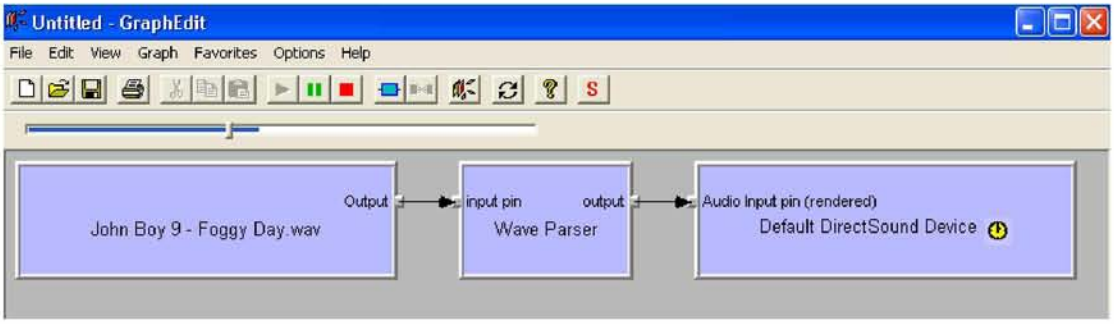


Figure 2-3. Rendering the WAV file

Enumerating Filter Types

To understand what kinds of filter graphs you can create in GraphEdit, you have to enumerate (list) all the possible filters available for DirectShow's use. From GraphEdit's Graph menu, select the first item, Insert Filters, and you'll see the dialog box shown in Figure 2-4.

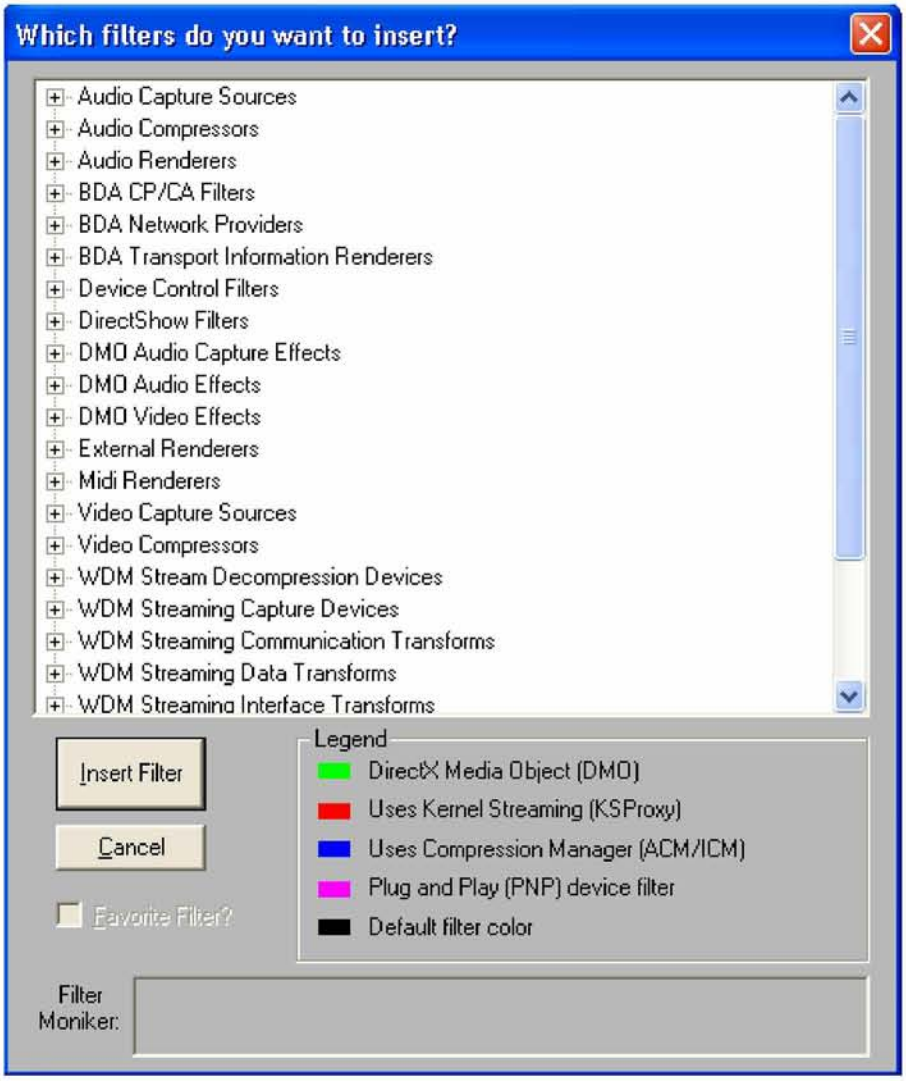


Figure 2-4. GraphEdit's Insert Filters dialog box enumerating all available DirectShow filters

This dialog box uses the Windows convention of expandable lists of items. To the left of each list entry in the dialog box is a plus sign, which hides a list of DirectShow filters underneath. Clicking on the DirectShow Filters entry will present the entire list of DirectShow filters available to GraphEdit. The other filters enumerated in the Insert Filters dialog box are Windows Driver Model (WDM) devices (all valid WDM devices are available to DirectShow as filters), DirectX Media Objects (DMOs), Video for Windows devices, and so forth.

Building a Filter Graph from Scratch

Now that you've enumerated the filters available to DirectShow, it's possible to build up a filter graph—say, one that will render an AVI movie—from scratch. Begin by selecting New from the File menu, which will clear out any existing filter graph. Next you'll need a source filter that points to the AVI file. From the list of DirectShow filters, select the entry labeled File Source (Async) and click the button marked Insert Filter. You'll immediately be presented with a file selection dialog box with the message "Select an input file for this filter to use." Select the AVI file Sunset.avi from the CD-ROM, and you should see the filter within GraphEdit, as shown in Figure 2-5.

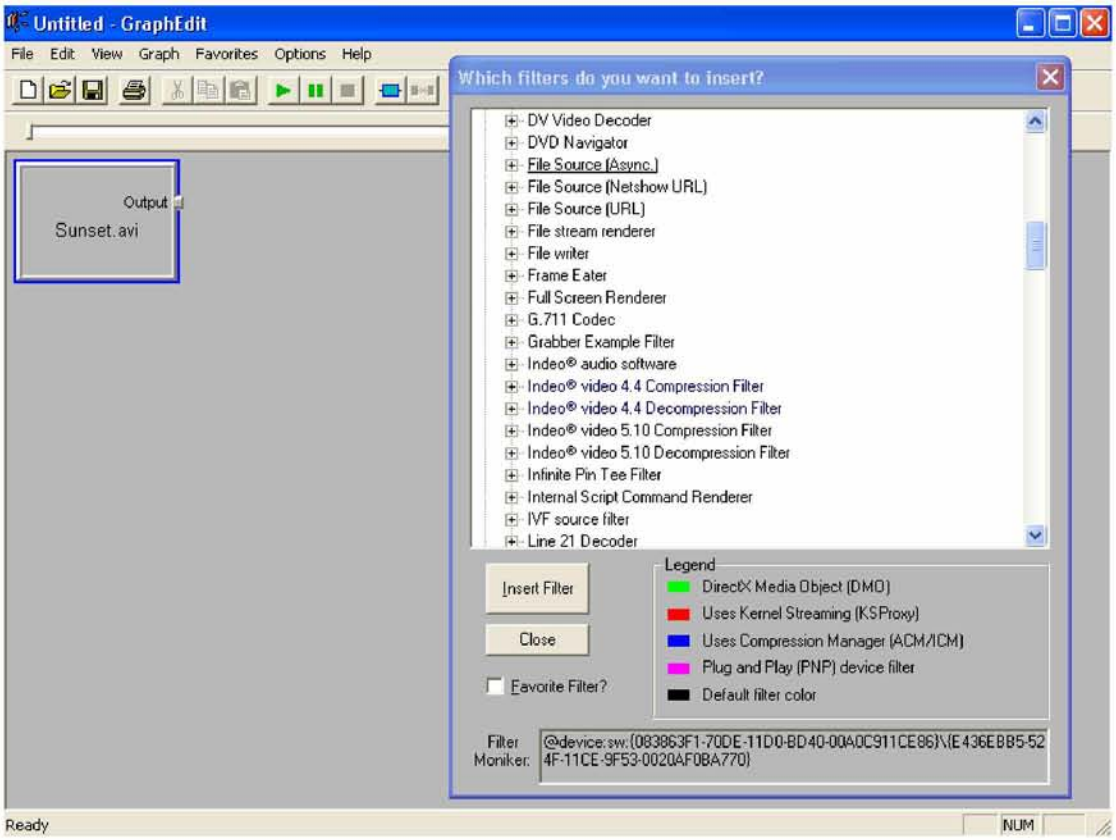


Figure 2-5. A source filter—the beginning of every DirectShow filter graph

The AVI file in the source filter needs to be divided into two streams, one with video data and one with audio data. The DirectShow transform filter AVI Splitter makes the division; insert it into the filter graph.

The source filter and the transform filter need to be connected, from the output pin of the source filter to the input pin of the transform filter. To make this connection, click the output pin of the source filter, drag the mouse pointer over the input pin of the transform filter, and release the mouse button, as shown in Figure 2-6.

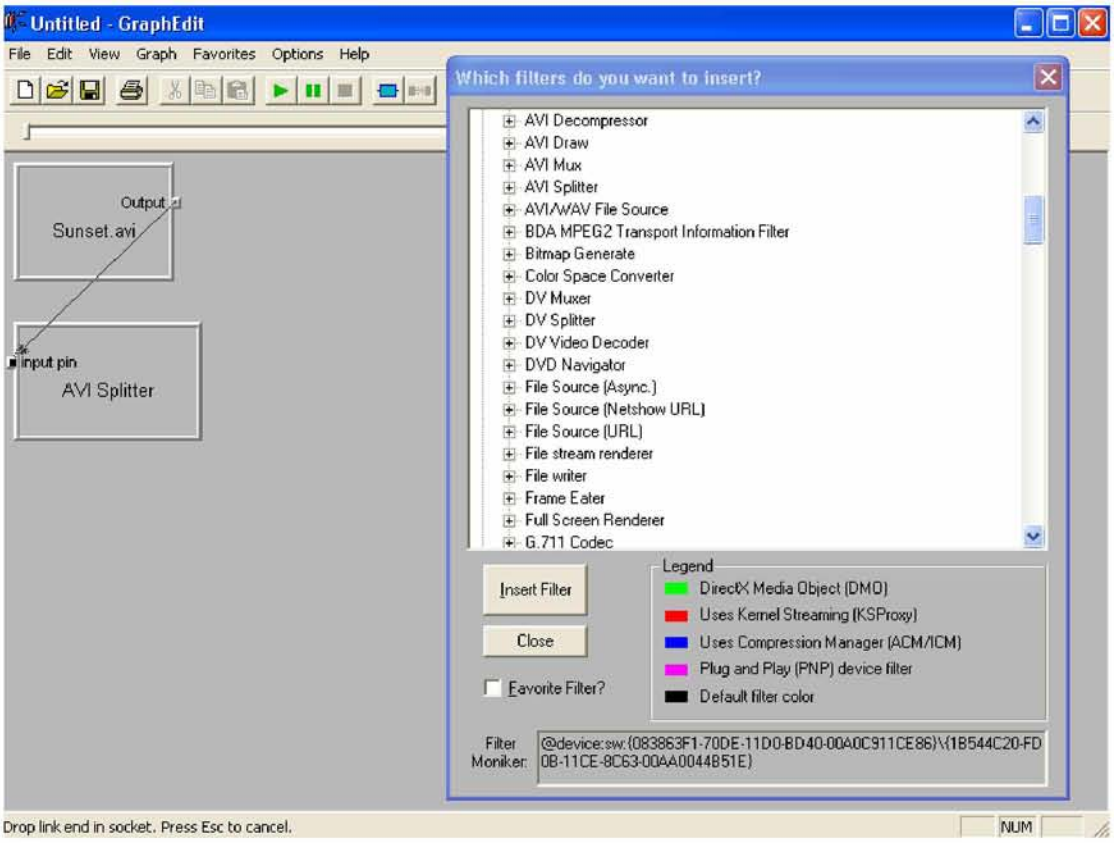


Figure 2-6. Creating a connection between an output pin and an input pin

GraphEdit automatically rearranges the filters so that they fit together smoothly, as Figure 2-7 shows.

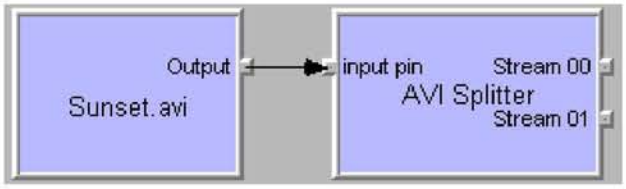


Figure 2-7. GraphEdit adjusting the position of the filter

The AVI Splitter transform filter produces two streams, which are available on output pins Stream 00 and Stream 01. Although it's less than clear from these labels, Stream 00 is a video stream and Stream 01 is an audio stream. (This confusing nomenclature is a result of the fact that an AVI file can have a wide variety of media types stored within, which means you couldn't appropriately name a pin VideoOut or AudioOut because there might be no video or audio in any given AVI file.) To render the audio stream, you need to insert the renderer filter Default DirectSound Device from the list of Audio Renderers in the Insert Filters dialog box. Connect output pin Stream 01 to the Audio Input pin of the renderer filter. Now your filter graph should look like the one shown in Figure 2-8.

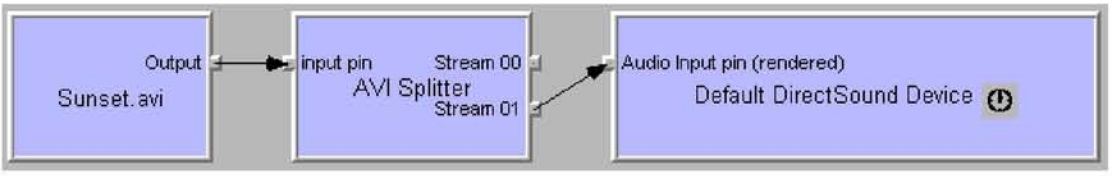


Figure 2-8. A DirectSound renderer filter added

Rendering the video is a two-step process. Because this AVI file contains digital video (DV) and audio data, a transform filter known as the DV Video Decoder must be added to the filter graph. Add the filter, and connect pin Stream 00 from the AVI Splitter filter to the XForm In pin of the DV Video Decoder. Finally a video renderer can be selected from the list of DirectShow filters. Although several types of video renderers are available—including full-screen and off-screen renderers—we'll use the generic Video Renderer, which sends the video to a window on the display that will be opened when the filter graph begins to run. Add the filter, and connect the output pin XForm Out from the DV Video Decoder to the Input pin of the Video Renderer. When it's all done, the filter graph will look like Figure 2-9.

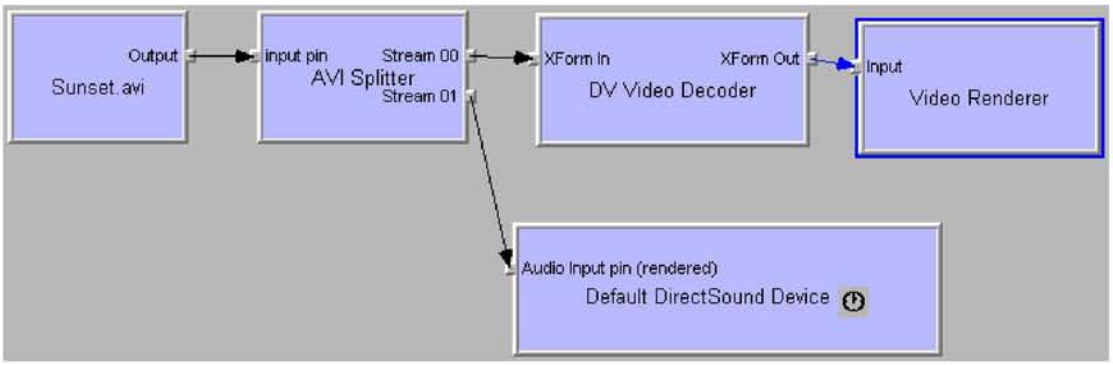


Figure 2-9. A complete filter graph rendering both audio and video streams of the AVI file

At this point, click Play. A new window titled ActiveMovie Window will open on the screen and display the AVI file as it plays, as shown in Figure 2-10. The window's title bar is a holdover from the time, a few years back, when DirectShow was known as ActiveMovie. (The movie has a silent soundtrack, so don't expect to hear anything from the speakers as the movie is playing.)

That's what it takes to create a filter graph from scratch; filters have to be inserted into the filter graph manually and connected appropriately. However, GraphEdit provides two alternative techniques to speed the creation of filter graphs: rendering pins and Intelligent Connect.

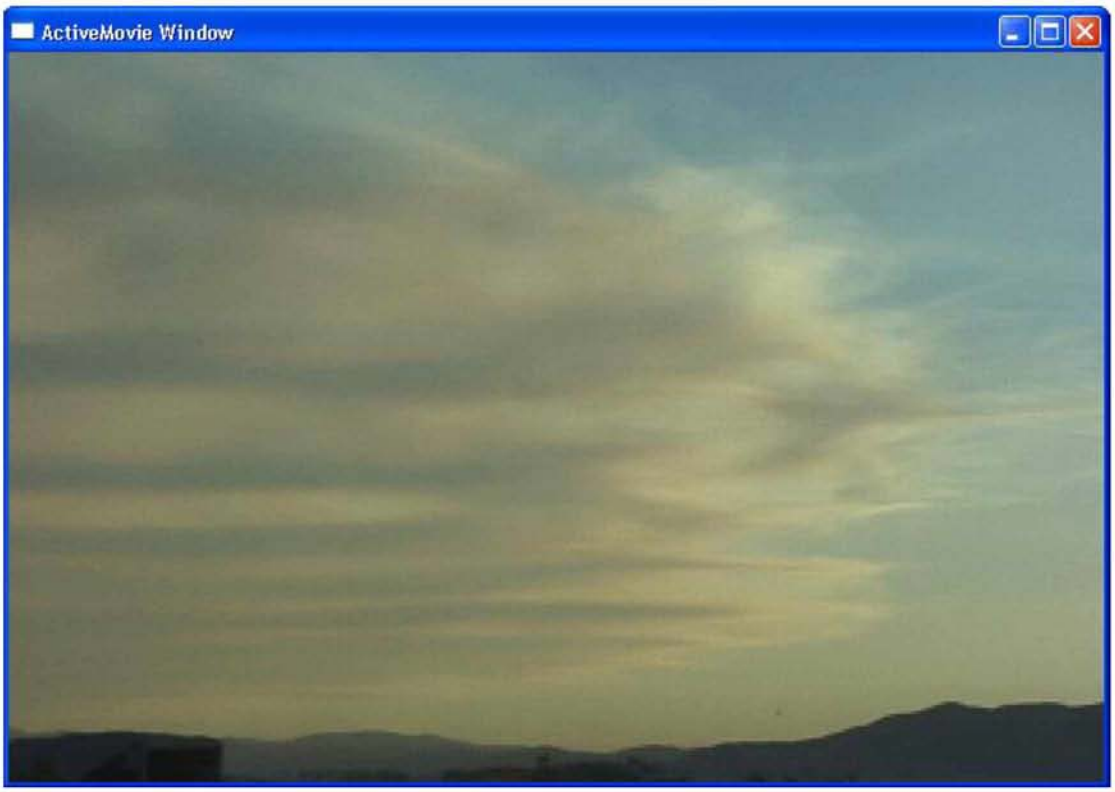


Figure 2-10. The AVI file in mid-playback

Creating Filter Graphs with Rendering Pins

Another technique that quickly renders media files from within GraphEdit uses rendering pins. Starting from an empty filter graph, add a File Source (Async) source filter to the filter graph and select Sunset.avi as the input file. Right-click the output pin of the filter. You'll see the menu shown in Figure 2-11.

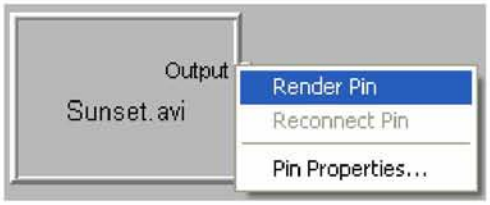


Figure 2-11. The pin menu with per-pin options on each filter

Select Render Pin. The resulting filter graph should be identical to the filter graph you built from scratch—without the additional steps of finding the correct DirectShow filters, inserting them into the filter graph, and connecting them appropriately. You can "Render Pin" from any output pin in the filter graph, not

just a source filter. You can use this feature to finish a partially complete filter graph or, as in this case, to build the entire filter graph.

Simplifying Design Tasks with Intelligent Connect

In Chapter 1, you used the filter graph to connect output pins that accepted different media types by stringing a set of intermediate transform filters between the pins. This feature, known as Intelligent Connect, allows the filter graph to assume most of the heavy lifting involved in graph building. This same feature can be put to work in your DirectShow programs to simplify coding tasks in many situations. Intelligent Connect is turned on in GraphEdit by default. (To turn it off, clear the Connect Intelligent menu option in the Graph menu.)

To use Intelligent Connect, begin by creating a filter graph with two filters: first a File Source (Async) filter, pointing to the file Sunset.avi, and next a Video Renderer filter. You've now got a filter graph with a source filter and a renderer filter, but no transform filters. However, GraphEdit will let you create a connection from the output pin of the source filter to the input pin of the renderer filter, as shown in Figure 2-12.

Figure 2-12. A source filter can't be connected directly to a renderer filter, unless…

This connection won't work on its own. The media type of the output pin of the source filter does not agree with the media type required by the input pin of the video renderer. That's when Intelligent Connect steps in and adds the intermediate filters needed to transform AVI file data into video that can be rendered to the display, as shown in Figure 2-13.

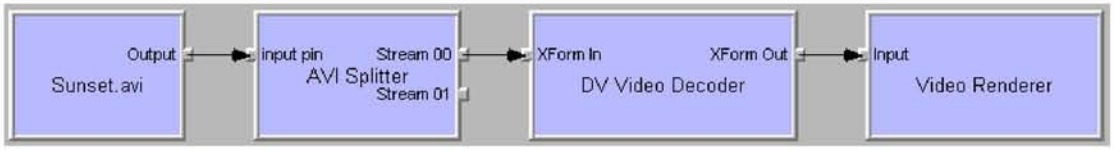
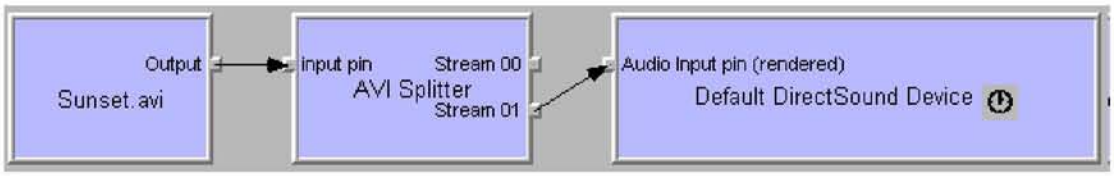


Figure 2-13. …Intelligent Connect is used, in which case intermediate transform filters are added.

The filter graph has added AVI Splitter and DV Video Decoder transform filters, both of which are needed to translate the output of the source filter into video data that can be rendered to the display. Note that the audio stream hasn't been rendered. Rather than rendering the media file, Intelligent Connect simply

found a path to connect a source filter to a renderer filter. If the renderer filter had been a Default DirectSound Device rather than a Video Renderer, the resulting



filter graph would have looked like Figure 2-14.

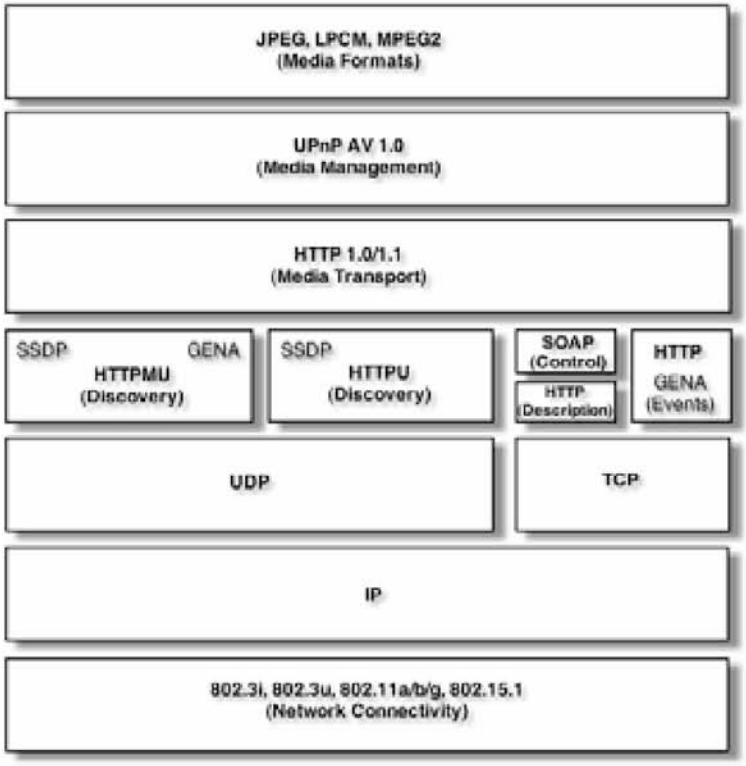
Figure 2-14. A source filter connected to an audio renderer requiring one transform filter

D.Eastern Europe Broadcast Assignments

```
F_EEU_BROAD RCDATA
BEGIN
    1L,          // Lowest index
    69L,         // Highest index
    49750000L,   // 1
    59250000L,   // 2
    77250000L,   // 3
    85250000L,   // 4
    93250000L,   // 5
   175250000L,   // 6
   183250000L,   // 7
   191250000L,   // 8
   199250000L,   // 9
   207250000L,   // 10
   215250000L,   // 11
   223250000L,   // 12
        0L,   // 13 Not used
        0L,   // 14 Not used
        0L,   // 15 Not used
        0L,   // 16 Not used
        0L,   // 17 Not used
        0L,   // 18 Not used
        0L,   // 19 Not used
        0L,   // 20 Not used
   471250000L,   // 21
   479250000L,   // 22
   487250000L,   // 23
   495250000L,   // 24
   503250000L,   // 25
   511250000L,   // 26
   519250000L,   // 27
   527250000L,   // 28
```

```
    535250000L,     // 29
    543250000L,     // 30
    551250000L,     // 31
    559250000L,     // 32
    567250000L,     // 33
    575250000L,     // 34
    583250000L,     // 35
    591250000L,     // 36
    599250000L,     // 37
    607250000L,     // 38
    615250000L,     // 39
    623250000L,     // 40
    631250000L,     // 41
    639250000L,     // 42
    647250000L,     // 43
    655250000L,     // 44
    663250000L,     // 45
    671250000L,     // 46
    679250000L,     // 47
    687250000L,     // 48
    695250000L,     // 49
    703250000L,     // 50
    711250000L,     // 51
    719250000L,     // 52
    727250000L,     // 53
    735250000L,     // 54
    743250000L,     // 55
    751250000L,     // 56
    759250000L,     // 57
    767250000L,     // 58
    775250000L,     // 59
    783250000L,     // 60
    791250000L,     // 61
    799250000L,     // 62
    807250000L,     // 63
    815250000L,     // 64
    823250000L,     // 65
    831250000L,     // 66
    839250000L,     // 67
    847250000L,     // 68
    855250000L,     // 69
END
```

E. UPNP/DLNA – ARCHITECTURE



Media Formats

Media Formats describe how digital content is encoded and formatted for each of the three classes of media: image, audio, and AV. The media format profiles are very explicit, with attributes, parameters, system, and compression level details defined in sufficient detail ensure interoperability between DLNA compliant devices. The current baseline set of media formats that each specific device type must support are listed in the table below.

In addition to the baseline media formats that must be supported, DLNA also defines a set of optional media formats and provides specific rules about using optional formats between compatible devices and conversion between optional and mandatory formats.

The focus on media formats is a dividing line between the work of the UPnP Forum and DLNA. The UPnP Forum focused on achieving device interoperability, which was accomplished. But the lack of prescribed media profiles prevented the UPnP architecture from delivering media interoperability and led to the founding of DLNA.

Media Transport

Media Transport defines how content moves across the network. DLNA devices that send or receive any media content to/from the network must support HTTP 1.1 (including chunked transfer encoding, persistent connections, and pipelining) as the baseline transport mechanism. In addition, Real-time Transport Protocol (RTP) is available as an optional media transport protocol. But, the mandatory requirement for HTTP 1.1 must always be met.

Media Management

Media management enables devices and applications to identify, manage, and distribute digital media content across network devices. The DLNA Guidelines incorporate the UPnP Forum AV and Printer technology as the basis for DLNA media management. The services provided by this technology are: Content Directory, Connection Manager, AV Transport, and Rendering Control.

Content Directory

The Content Directory service provides a mechanism for each content server on the network to provide a uniform directory of all its available content to any interested devices on the network. Every content server **must** have an instance of this service. This service might be used to display a list of songs stored on an MP3 player, still-images comprising various slide shows stored on a PC, movies stored in a DVD jukebox, TV shows currently being broadcast by a Set-top Box, songs stored in a CD Jukebox, TV programs that had been downloaded to a PVR, photos stored in a digital camera, and many more. Nearly any type of content can be listed via the Content Directory service, even for devices that support multiple types of content. The information about the content (metadata) returned by the Content Directory service includes properties such as its name, artist, creation date, size, etc. In addition, the metadata also indicates the transfer protocols and data formats that are supported for each piece of content on the server. This information is used by the Control Point to determine if a given Media Renderer is capable of rendering the content in its current format or if some type of transcoding is required.

Connection Manager

The Connection Manager service determines how the digital media content can be transferred between two devices on the network. Each device that sends or receives content **must** implement the Connection Manager service. This service provides a mechanism for devices to:

Match capabilities between server and render devices.

• Set up and tear down connections between devices.

• Discover information about current transfers in the network.

When making connections, the Connection Manager service is the interface between the devices and the TCP/IP stack.

AV Transport

The AV transport service enables control over the "playback" of audio and video streams including the ability to Stop, Pause, Seek, etc. This service type defines a common model for AV transport control suitable for a generic user interface. It can be used to control a wide variety of disc, tape, and solid-state media devices such as CD players, VCRs, and MP3 players. Depending on the supported transfer protocols and data formats, this service may or may not be implemented. Although most media will be sent across the network as data it may be more efficient to transfer the media data stream using other means. An example is when a personal video recorder is the media source and a TV set is the media renderer (player). An Ethernet connection would not be as efficient as an s-video connection. Using a transfer medium that is not part of the TCP/IP network is called an **out of band transfer**. These transfers are not defined by the UPnP AV specification but are recommended and supported by the manufacturer of the media equipment.

Rendering Control

Most rendering devices contain a number of dynamically configurable attributes that affect how the current content is rendered. For example, video devices, such as TVs, allow user control of display characteristics such as brightness and contrast, while audio devices allow control of audio characteristics such as volume, balance, and equalizer settings. The Rendering Control service is intended to provide control points with the ability to query and/or adjust any rendering attribute that the device supports. The Rendering Control service enables a control point to:

• Discover the attributes supported by the device.
• Retrieve the current setting of any supported attribute.
• Change the setting of any modifiable attribute.
• Restore the settings defined by a named preset.

Device Discovery and Control

Device discovery and control enables a device on the home network to discover the presence and capabilities of other devices on the network and collaborate with these devices in a uniform and consistent manner. DLNA incorporates the UPNP Forum Device Architecture 1.0 as the basis for its device discovery and control.

Discovery

Device discovery is the first step in UPnP networking. When a new device is added to a network, the UPnP discovery protocol (SSDP) allows the device to advertise its services to all control points on the network via a multicast. Similarly, when a new control point is added to the network, SSDP allows the control point to search for devices of interest using a multicast. Thus, by listening to the standard multicast address, control points and devices can be made aware of new services being offered on the network and respond to service requests. In each case, the response is a discovery message that contains a small number of essential specifics about the device or its services, e.g., its UPnP type, its universally unique identifier (UUID), and a URL to more detailed information. When a control point discovers a new device or service of interest – either as a result of a device's advertisement or a specific search – the control point must then use the URL in the discovery message to retrieve a description of the device and its capabilities.

Device Description

A UPnP device description includes vendor-specific manufacturer information like the model name and number, serial number, manufacturer name, URLs to vendor-specific Web sites, and a URL for presentation. For each service included in the device, the device description lists the service type, name, a URL for a service description, a URL for control, and a URL for eventing. A device description also includes a description of all embedded devices. Note that a single physical device may include multiple logical devices. A UPnP device description is written by the device vendor, and is expressed in XML syntax and based on a standard UPnP Device Template produced by the UPnP Forum working committee.

Service Description

A UPnP service description includes a list of commands, or *actions*, the service responds to, and parameters, or *arguments*, for each action. A service description also includes a list of variables. These variables model the state of the service at run time, and are described in terms of their data type, range, and event characteristics. Like a UPnP device description, a UPnP service description is written by a UPnP vendor. The description is in XML syntax and is based on the standard UPnP Service Template. Retrieving a UPnP device description is straight forward: the control point issues an HTTP GET request on the URL in the discovery message, and the device returns the device description. Retrieving a UPnP service description is a similar process that uses a URL within the device description.

Control

Once the device and its service descriptions are retrieved, a control point can ask those services to invoke actions and the control point can poll those services for the values of their state variables. Invoking actions is a form of remote procedure call; a

control point sends the action to the device's service, and when the action has completed (or failed), the service returns any results or errors. Polling for the value of state variables is a special case of this scenario where the action and its results are predefined. To control a device, a control point invokes an action on the device's service. To do this, a control point sends a suitable control message to the control URL for the service (provided as part of the service element in the device description). These messages are encapsulated in SOAP, and sent via HTTP requests. In response, the service returns any results or errors from the action – again using SOAP and HTTP. The effects of the action, if any, may also be reflected by changes in the variables that describe the run-time state of the service. When these state variables change, events are published to all interested control points. To determine the current value of a state variable, a control point may poll the service. Similar to invoking an action, a control point sends a suitable query message to the control URL for the service. In response, the service provides the value of the variable; each service is responsible for keeping its state table consistent so control points can poll and receive meaningful values As long as the discovery advertisements from a device have not expired, a control point may assume that the device and its services are available. If a device cancels its advertisements, a control point must assume the device and its services are no longer available.

Eventing

As the section on Description explains, a UPnP service description includes a list of actions the service responds to and a list of variables that model the state of the service at run time. If one or more of these state variables are evented, then the service publishes updates when these variables change, and a control point may subscribe to receive this information. Throughout this section, *publisher* refers to the source of the events (typically a device's service), and *subscriber* refers to the destination of events (typically a control point). Both subscriber and publisher messages are delivered via a version of HTTP that has been extended using General Event Notification Architecture (GENA) methods and headers developed by the UPnP Forum. To start receiving events, a subscriber sends a *subscription message*. If the subscription is accepted, the publisher responds with a duration for the subscription. To keep the subscription active, a subscriber must renew its subscription before the subscription expires. When a subscriber no longer needs eventing from a publisher, the subscribercancels its subscription. The publisher notes changes to state variables by sending *event messages*. Event messages contain the names of one of more state variables and the current value of those variables, expressed in XML. A special *initial event message* is sent when a subscriber first subscribes; this event message contains the names and values for all evented variables and allows the subscriber to initialize its model of the state of the service. To support scenarios with multiple control points, eventing is designed to keep all subscribers equally informed about the effects of any action. Therefore, all subscribers are sent all event messages, subscribers receive event messages for all

evented variables (not just some), and event messages are sent no matter why the state variable changed (either in response to a requested action or because the state the service is modeling changed). An example of this would be the volume control on digital TV that was a DMP/DMR combined device. Turning the physical volume control on the device would send an event message to the network with the new volume level. Conversely, a DMC could send a volume action to the TV that would result in it changing its volume and sending an event message to the network to that effect. Some state variables may change value too rapidly for eventing to be useful. One alternative is to filter, or moderate, the number of event messages sent due to changes in a variable's value. Some state variables may contain values too large for eventing to be useful; for this, or other reasons, a service may designate one or more state variables as *non evented* and never send event messages to subscribers. To determine the current value for such non-evented variables, control points must poll the service explicitly

Presentation

Presentation exposes an HTML-based user interface for controlling/monitoring the status of a device. It augments the standard UPnP control and eventing mechanisms by providing a browser based avenue for sending actions to a device and receiving notification of state changes from a device. The URL for the presentation page is provided by the device description. To retrieve the presentation page, an HTTP GET request is issued to the presentation URL and the device returns the presentation page. Loading the page into a browser will, provided the page enables it, allow the user to control the device or view its status. Unlike the interactions defined by the UPnP Device and Service Templates, the capabilities of the presentation page are completely specified by the device vendor. The implementation details are also up to the device vendor, but the use of an embedded web server in conjunction with the device's native capabilities is the most common method of implementing this feature.

Network Stack

The basis for UPnP Networking (and thus DLNA) is the TCP/IP v4 protocol. Every device must implement a DHCP client, and search for a DHCP server when first connected to the network. If a DHCP server is discovered, the device must use the IP address assigned by the server. If no DHCP server is discovered, the device must use Auto-IP to generate a link-local IP address. Auto-IP uses an implementation dependent algorithm to generate an address in the 169.254/16 range. The first and last 256 addresses in this range are reserved and must not be used. After developing an address, the device must determine if the address is available by using an ARP probe. If the device receives a response, the address is assumed to be in use and the device must generate and test a new IP address. A device that has configured via Auto-IP must periodically check for the presence of a DHCP server. If a DHCP server is discovered, the device must switch to the IP address allocated to it by the

DHCP server. In order to switch between IP addresses, the device must cancel any outstanding UPnP Discovery advertisements and re-issue them under the new address. In addition to IP addressing, UPnP makes extensive use of both the UDP and TCP protocols. Discovery is done via an HTTP Multicast over UDP and is used by devices to advertise their presence to the network and by control points to discover what devices exist on the network. Definition, control, and eventing services are delivered via HTTP over TCP.

Network Connectivity

Three network connection technologies are incorporated in the DLNA 1.5 Interoperability Guidelines: 10Base-T and 100Base-T Ethernet (802.3i / 802.3u) for wired connections, WiFi (802.11a /802.11b /802.11g) for wireless connections, and Bluetooth for wireless connections for mobile handheld devices such as cell phones and PDAs. Additional network connections such as 1000Base-T Ethernet (802.3ab) and faster WiFi (802.11n) will be added to the Guidelines in the future. It should also be noted that many other networking technologies such as LonWorks, CeBus, X-10, and Universal Powerline Bus (UPB) could be supported via UPnP Bridges.

Intel Device Validator

With the push of a few buttons, Device Validator automatically tests devices for UPnP and UPnP AV compliance. When testing is complete, the UI indicates test results by one of three light states (see Figure 1, page 68).

- Green Light: Pass—complies with pertinent UPnP standards
- Red Light: Fail—specific points of non-compliance are identified
- Yellow Light: Warning—areas of technically compliant behavior, but not recommended behavior

In addition to checking for compliance with the UPnP specification, Device Validator will warn the developer when best known practices and implementation guidelines[1] are not followed. Another useful feature of Device Validator is its plug-in architecture that allows the addition of new test modules. This makes Device Validator a valuable tool for adding new test functionality to a regression harness.

[1] The official UPnP Vendor's Implementation Guide can be found at *http://upnp.org/resources/documents.asp*.

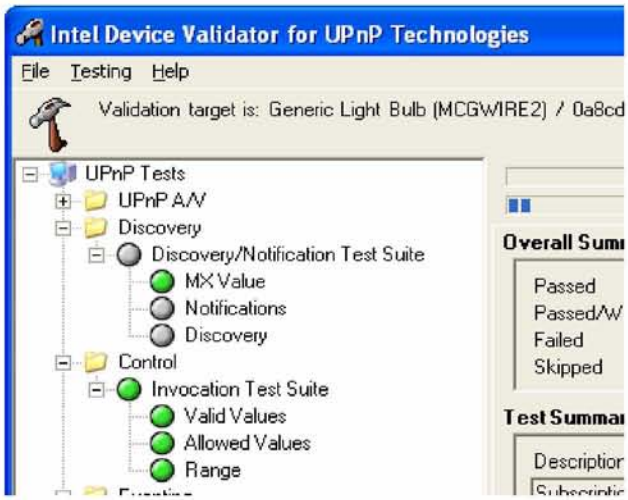


Figure 1. The GUI for Intel Device Validator

Intel UPnP Tools

In addition to development and validation tools, Intel has provided developers with tools for manual testing, including a few tools that can work with any type of UPnP device. This suite of tools is available at:
http://www.intel.com/labs/connectivity/upnp/

Intel Device Spy

Intel Device Spy is a universal control point (UCP) that supports manual diagnostics for individual actions and events.

Figure 2. The GUI for Intel Device Spy.

Device Spy offers a developer these capabilities:

- Detection of all UPnP devices on a network
- Display of detailed UPnP device information
- Action invocation

- Event monitoring
- Error reporting

Device Spy is an excellent learning tool for developers new to UPnP technology. It provides an opportunity to interact with UPnP devices and begin to understand how such devices work.
For more experienced developers, Device Spy provides a way to control any type of UPnP device. It allows them to manually test features during implementation-debug cycles, or to examine how existing UPnP devices work. Through its invocation stress testing, error reporting, and outbound/inbound packet tracing, Device Spy enables developers to discover and fix potential problems that may occur during UPnP software development.