Faq MHW

Created by Tideglo

Indice della sezione Pantalk

Come noterete ogni funzione presenta il nome effettivo e il nome simbolico. Questo perchè ognuno può associare ad ogni funzione il nome simbolico che vuole.
Spesso nella descrizione delle funzioni cercherò di fare riferimento anche alle funzioni che sono simili o della stessa categoria della funzione presa in considerazione, in questo modo sarà più facile vedere tutte le caratteristiche e capire le varie funzioni.

Sommario

Introduzione

In questo paragrafo vorrei fare solo un'introduzione a quello che dovrebbe/potrebbe/dovrà essere questo documento!

Lo stile che assumerà questo documento sarà "assomigliante" a una sorta di blog/lezioni (anche se non mi piace chiamarle così) su quello che potrebbe aiutarci a capire meglio il linguaggio Pantalk. Ovviamente i vostri commenti, nonchè suggerimenti o critiche riguardo una spiegazione di una qualsiasi cosa, o magari anche solo un'aggiunta di informazioni, è sempre una cosa ben accetta,

quindi non esitate a commentare/aiutare... Tengo infine a precisare che io non mi ritengo assolutamente un professore (e tanto meno lo sono), quindi tutto quello che verrà scritto qui è da prendere come roba scritta da un utente normale e non da uno che fa di lavoro l'esplicatore. Visto che non sono nemmeno un professore non ho nemmeno intenzione di dare lezioni private a nessuno, semplicemente se qualcuno pone una domanda e mi andrà/saprò rispondere lo farò, ma mai mi sentirò in obbligo a rispondere a ogni domanda che farete!

Questo documento vuole essere una raccolta di informazioni riguardanti il pantalk, così potrò rileggermele, approfondirle e magari passarle ai posteri senza che loro debbano reiniziare da zero come ho fatto io (si fa parecchia fatica!).

Riguardo agli argomenti trattati:
- si potrà discutere di tutto ciò che riguarda il pantalk e come è utilizzato nel mhw.
- qualsiasi cosa che verrà considerata materiale anche solo lontanamente illegale verrà immediatamente cancellato senza possibilità di replica.

Dopo questa "lunga" premessa passiamo ora ad esplicare cosa è il pantalk!!
Tanti anni fa.... negli anni '90 per la precisione ci fu una ditta (la hyperpanel) che decise di creare un linguaggio che potesse essere utilizzato su diverse piattaforme senza per questo dover riscrivere ogni volta, ricompilare il programma e adattarlo alla piattaforma. Oltre a questo doveva anche saper aiutare in modo spropositato i creatori di programmi per questo linguaggio rendendolo il più semplice possibile.
Ebbene si, se state pensando a un linguaggio che possa essere eseguito su diverse piattaforme senza dover essere ricompilato non potete non pensare al famoso e ormai già confermato JAVA.
Come per il JAVA, il Pantalk è costituito da una Virtual Machine che provvede a eseguire sulla piattaforma il programma scritto in Pantalk.
Ma perchè studiarsi il Pantalk? Non bastava il Java? Si bastava il java eccome! Solo che il java è stato inventato principalmente per poter essere eseguito sui computer e non su piattaforme completamente diverse a livello hardware. Un esempio ne sono gli STB. Principalmente il pantalk viene usato per scrivere i firmware dei vari decoder, ma non solo, tempo fa avevo anche visto dei cellulari che utilizzavano il pantalk.
C'è anche da aggiungere che il pantalk rispetto al Java è molto più semplice per quel che riguarda la gestione di eventi e molte altre piccole cose... insomma... non è stato inventato per complicare le cose, ma perchè se ne sentiva la necessità.
Attenzione perchè nella sua semplicità a volte il pantalk può risultare ostico fino a quando non se ne capisce bene a modo il meccanismo generale, soprattutto ricordiamoci che c'è scarsa, se non zero, documentazione a riguardo... uno dei motivi è che questo linguaggio è utilizzato dagli STB che rende il tutto molto più misterioso.

Principalmente noi quando parleremo del pantalk ci riferiremo a quello che è utilizzato sui vari decoder.
Perchè questa differenziazione? Beh perchè il pantalk è il linguaggio utilizzato ma la struttura di un programma scritta nel decoder è diversa da quella scritta per altre piattaforme standard (dove si intende il pantalk classico).
Pian piano scopriremo alcune differenze e impareremo ad utilizzare i tool più complessi per poter creare veri e propri firmware (non è mia intenzione seguire la costruzione da zero di un firmware, ma solo di esplicarne le caratteristiche).

Dimenticavo quasi di dire una cosa... tutti i decoder che utilizzano il pantalk (almeno quelli che conosco io) sono denominati GoldBox (questo nome è presente nella descrizione del decoder stesso) perchè sfruttano una struttura firmware che gli permette di eseguire i programmi in pantalk per STB ed erano stati introdotti con l'arrivo del sistema di codifica seca. Se non ci avete mai fatto caso, potrete notare che sia i firmware (originali) dei goldbox pioneer (qualsiasi modello), philips (qualsiasi modello), strong, sony, kenwood e nokia hanno tutti la stessa struttura dei menu, la stessa grafica (forse cambia qualche colore! In alcuni è su base blu altri su base rossa) e persino stesse funzioni! (ovviamente sono tutti decoder con cam base seca)
In questo modo un provider poteva adattare il firmware alla sua emittente per tutti i goldbox tramite un semplice aggiornamento OnAir (via sat) senza dover stare a scrivere migliaia di codice per decine di decoder diversi!

Vabbeh... credo di aver detto tutto quanto... almeno per l'introduzione

MHW, GoldBox e firmware

Eccoci qui finalmente a cercare di spiegare cosa è il MHW, cosa sono i firmware e cosa è un GoldBox.

Il GoldBox è il decoder utilizzato con l'unico scopo di poter decriptare in base allo standard DVB tutti i messaggi che arrivano via satellite da una qualsiasi emittente TV.
Il GoldBox possiede al suo interno una cam seca non rimovibile e presenta una serie di specifiche che devono essere rispettate affinchè sia considerato un GoldBox. Prima tra tutte è la cam che deve essere assolutamente in base seca. Poi deve avere le cose più scontate per un ricevitore, tra le quali: un tuner (generalmente si ha anche l'uscita per poter collegare in cascata un altro decoder, ma non è necessario), due prese scart (una per l'uscita tv l'altra per il videoregistratore), una porta seriale, una porta parallela, le uscite Audio e audio digitale e per finire la cosa fondamentale, il modem 56k.

OT:
Ovviamente ci sono alcune eccezzioni, ad esempio il nokia 9701s è l'unico goldbox ad essere un goldbox e anche un decoder C.I. (common interface). Non per questo rimane una mezza schifezza per quel che riguarda l'implementazione CI (causa hardware e non software!) infatti alcune cam sono rognose da digerire ma ho poi notato che anche altri decoder prettamente CI hanno tali problemi...
Chiuso OT.

Detto ciò... un firmware che l'è? Eheh domanda gnocca... ma risposta non tanto semplice! Il firmware è un programma che serve per fare funzionare una periferica in modo adeguato, nel nostro caso il decoder. Però il decoder non è altro che un agglomerato di altre periferiche che sono state messe assieme per eseguire un unico scopo...
Il firmware dei decoder è un insieme di driver che eseguono tante piccole operazioni e che sono tutti gestiti tramite pantalk.
Lo scopo del firmware del decoder è far visualizzare alla tv il canale selezionato, quindi sarà compito del firmware decidere se il canale ha un certo Transponder, una certa frequenza se può essere utilizzato con l'abbonamento corrente o se è un canale FTA (Free To Air), ecc...

Ora è necessario esporre un altro concetto... il famigerato MHW.
Tutto quello che fa il firmware viene deciso in MHW (MediaHighWay).

Il firmware dei goldbox viene diviso in due grandi parti, la parte MHW e la parte HDL (HardwareDependantLayer).
L'HDL che in tanti cercano ed è molto acclamato al momento non ci interessa, ma spiegherò brevemente in che cosa consiste.
La parte HDL è la parte software che contiene i driver delle periferiche, che gestisce a basso livello tutto ciò che serve al decoder per poter lavorare correttamente, inoltre è presente sempre in HDL la virtual machine che interpreta il Pantalk.
Da qui possiamo associare l'HDL come il sistema operativo del nostro decoder che provvederà ad intercettare i vari eventi e a gestire tutto quello che gli chiediamo di fare!
In MHW invece noi troviamo tutto il nostro firmware scritto in Pantalk.
Ovviamente per MHW non si intende un file scritto in pantalk ma una struttura ben definita che deve sempre essere organizzata in un certo modo, in diversi file, moduli, per poter essere definita per l'appunto MHW.
Più avanti vedremo meglio come è costituito il MHW.

Piccola precisazione: il MHW ovviamente è la parte del firmware non disassemblabile per il tipo di processore che si ha, mentre l'HDL può essere disassemblato.
Ad esempio, sui philips e sui pioneer è stato utilizzato un processore ST20 quindi per disassemblare l'HDL sarà necessario andare a scartare il mhw e iniziare il disasm di tutto il resto del codice. Negli strong è stato usato un processore MIPS, nel nokia 9701s un ARM...
Il MHW è anch'esso disassemblabile, ma più che disassemblabile io direi decompilabile e interpretabile! Già perchè come andremo ad analizzare e a vedere, il MHW è compattato, compresso e compilato in pcode. Il pcode per chi non sa cosa sia, è un'associazione di byte ad ogni istruzione pantalk. Come avviene per il java! Per questo motivo è facile avere il codice sorgente di tutta la parte MHW.

La struttura MHW

Analizziamo la FAT MHW di un firmware....
Apriamo un firmware in modo binario, e cerchiamo la parola "boot".

Fate attenzione perchè spesso si trovano anche tre o quattro ricorrenze di boot mentre a noi interessa solo trovare la parola boot che si trova nella **ModuleList** (dopo spiegherò che cosa è). Come fare a sapere quando troviamo la boot della module list? Semplice, se dopo si trovano altri nomi tipo basic, alarm, antena, install, ecc... tutti separati dallo stesso numero di byte abbiamo trovato la module list!

esempio (nokia 9701s)

```
00152aa0h: 62 6F 6F 74 00 00 00 00 00 00 00 00 2C 11 07 8C ; boot........,..Œ
00152ab0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00152ac0h: 61 6C 61 72 6D 00 00 00 00 00 00 00 2C 11 0E AC ; alarm.......,..¬
00152ad0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00152ae0h: 61 6C 6C 63 61 6D 00 00 00 00 00 00 2C 11 42 8C ; allcam......,.BŒ
00152af0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00152b00h: 61 6E 74 65 6E 61 00 00 00 00 00 00 2C 11 5C 9C ; antena......,.\œ
00152b10h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00152b20h: 62 61 73 69 63 00 00 00 00 00 00 00 2C 11 63 AC ; basic.......,.c¬
00152b30h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
```

00152b40h: 63 61 72 64 65 76 74 00 00 00 00 00 2C 12 6F 24 ; cardevt.....,.o$
00152b50h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00152b60h: 63 61 72 64 6C 00 00 00 00 00 00 00 2C 12 72 A4 ; cardl.......,.r¤
00152b70h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00152b80h: 63 6F 6D 6D 75 6E 00 00 00 00 00 00 2C 12 80 E0 ; commun......,.€à
00152b90h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00152ba0h: 64 76 62 63 69 00 00 00 00 00 00 00 2C 13 75 10 ; dvbci.......,.u.
00152bb0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00152bc0h: 65 65 67 65 73 74 00 00 00 00 00 00 2C 13 96 60 ; eegest......,.–`
00152bd0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00152be0h: 65 65 70 72 6F 6D 00 00 00 00 00 00 2C 13 AA CC ; eeprom......,.ªÌ
00152bf0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00152c00h: 66 64 6C 6F 61 64 00 00 00 00 00 00 2C 13 B2 48 ; fdload......,.²H
00152c10h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00152c20h: 6D 6F 64 63 6F 64 65 00 00 00 00 00 2C 13 B8 8C ; modcode.....,.¸Œ
00152c30h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00152c40h: 6D 6F 64 65 6D 00 00 00 00 00 00 00 2C 13 BE B0 ; modem.......,.¾°
00152c50h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00152c60h: 70 67 61 63 63 00 00 00 00 00 00 00 2C 13 D9 38 ; pgacc.......,.Ù8
00152c70h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00152c80h: 70 67 63 6F 6D 00 00 00 00 00 00 00 2C 13 ED 44 ; pgcom.......,.íD
00152c90h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00152ca0h: 70 67 63 6F 6D 62 00 00 00 00 00 00 2C 14 1D 08 ; pgcomb......,...
00152cb0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00152cc0h: 70 67 65 6E 72 00 00 00 00 00 00 00 2C 14 33 40 ; pgenr.......,.3@
00152cd0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00152ce0h: 70 67 67 72 69 6C 00 00 00 00 00 00 2C 14 40 78 ; pggril......,.@x
00152cf0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00152d00h: 70 67 6C 73 74 00 00 00 00 00 00 00 2C 14 60 40 ; pglst.......,.`@
00152d10h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00152d20h: 70 67 74 72 69 00 00 00 00 00 00 00 2C 14 86 0C ; pgtri.......,.†.
00152d30h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00152d40h: 70 69 63 74 75 72 65 00 00 00 00 00 2C 14 98 58 ; picture.....,.˜X
00152d50h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00152d60h: 72 61 64 69 6F 00 00 00 00 00 00 00 2C 14 A7 00 ; radio.......,.§.
00152d70h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00152d80h: 73 65 74 75 70 00 00 00 00 00 00 00 2C 14 EE 60 ; setup.......,.î`
00152d90h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00152da0h: 73 5F 63 68 6E 6C 00 00 00 00 00 00 2C 15 C2 08 ; s_chnl......,.Â.
00152db0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00152dc0h: 73 5F 63 6F 6E 66 69 67 00 00 00 00 2C 15 C9 2C ; s_config....,.É,
00152dd0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00152de0h: 73 5F 69 6E 66 6F 00 00 00 00 00 00 2C 15 E2 40 ; s_info......,.â@
00152df0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00152e00h: 73 5F 72 65 63 6F 72 64 00 00 00 00 2C 15 EA 58 ; s_record....,.êX
00152e10h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00152e20h: 73 5F 73 63 61 6E 00 00 00 00 00 00 2C 15 F2 78 ; s_scan......,.òx
00152e30h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....

00152e40h: 74 69 6D 65 00 00 00 00 00 00 00 00 2C 16 16 B8 ; time.....,..¸
00152e50h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00152e60h: 77 6B 5F 6D 6F 64 65 6D 00 00 00 00 2C 16 27 50 ; wk_modem....,.'P
00152e70h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....

Questo è un esempio di module list, state attenti perchè i moduli non sono sempre gli stessi, variano da firmware a firmware e nemmeno il numero degli stessi è sempre uguale, insomma c'è parecchia differenza tra firmware a firmware anche se generalmente mantengono la stessa struttura (nei firmware philips e pioneer si trovano spesso i moduli install, pers_*, invece qui non ci sono!)

Ora che abbiamo visto la module list possiamo iniziare a fare una breve analisi.
Come vedete si presenta come un listato di nomi seguito da numeri.
Ebbene si è proprio come si presenta, ossia un listato di nomi seguito da numeri!!!
In questo caso la module list ha ogni record costituito da 32 bytes. I primi 12 bytes sono usati per il nome del modulo (il nome finisce quando incontra il byte a 0 oppure se si sono raggiunti i 12 caratteri di lunghezza massima), i successivi quattro per indicare la posizione in memoria del modulo (ossia dei dati contenuti nel modulo, insomma, i file che contiene!), questa posizione è da leggere in esadecimale.
Gli altri sedici bytes sono inutilizzati e sono tutti a 0x00 (bytes di separazione).
Questa è la tipica struttura di una Module List, ma state attenti perchè ne esistono anche di soli 16 bytes (non ci sono i 16 bytes a 0x00 di separazione), oppure invece che avere l'indirizzo scritto in big-endian, lo hanno in little-endian (i byte sono da leggere all'incontrario...), ma è sempre possibile riconoscerla per il listato di moduli e perchè mantiene sempre la struttura: nome, posizione dati.

Ora, facciamo un passo indietro... guardiamo la stringa boot, ossia il record del modulo boot. Prima di questa stringa ci sono 4 bytes, che cosa sono? Sono il numero di moduli che la Module List contiene! Semplice no?!
Ebbene, questo generalmente rappresenta anche il numero massimo di moduli che il firmware può contenere, ossia se moltiplichiamo il NumeroMax di moduli per la lunghezza del record della module list (nel nostro caso 32) otteniano la lunghezza totale della module list in byte e la fine della module list stessa!

Perchè cercare boot? Perchè non cerchiamo pastasciutta? Beh... il modulo boot è il modulo che viene chiamato di avvio (eccerto se no lo chiamavano in altro modo) e per questo motivo deve sempre essere presente e deve sempre essere il primo nella lista dei moduli!!

Nella modulelist i riferimenti in memoria sono presi rispetto all'indirizzo in cui è situato il firmware, spesso in ram ma a volte anche in flash... nel nostro caso l'indirizzo base è 0x2C020000, sottraendolo ad ogni indirizzo troveremo il rispettivo offset nel firmware. Nei philips, nei pioneer e negli strong l'indirizzo è preso in base all'indirizzo in memoria Ram (quindi tocca studiare l'allocazione del firmware in ram, ma non è cosa complessa. Il più complesso è lo strong, tra tutti quelli che ho analizzato io, praticamente tutti i firmware esistenti).

Analizziamo ora quello che si trova subito dopo la ModuleList...

00152e80h: 00 00 00 09 00 00 00 35 00 00 00 18 00 00 00 09 ; ....5........
00152e90h: 00 00 00 A4 00 00 00 06 00 00 00 0B 00 00 00 14 ; ...¤............

00152ea0h: 00 00 00 1C 00 00 00 1C 00 00 00 07 00 00 00 05 ; .....
00152eb0h: 00 00 00 06 00 00 00 16 00 00 00 18 00 00 00 1B ; .....
00152ec0h: 00 00 00 13 00 00 00 0B 00 00 00 0F 00 00 00 12 ; .....
00152ed0h: 00 00 00 0D 00 00 00 0E 00 00 00 05 00 00 00 45 ; .....E
00152ee0h: 00 00 00 07 00 00 00 1A 00 00 00 08 00 00 00 0C ; .....
00152ef0h: 00 00 00 14 00 00 00 0F 00 00 00 08 ; .....

Eheh... semplice pure questo! Anche qui i valori sono scritti in Big-endian, ma nei decoder con base little saranno in little-endian quindi ad ognuno la sua struttura.
Come vedete ci sono 4 byte, questi quattro byte rappresentano un numero intero che va a identificare il numero di file che sono contenuti nel modulo rispettivo.

Esempio:
00 00 00 09 indica che ci sono 9 file contenuti nel modulo boot, ovviamente il valore è espresso in esadecimale

Avendo 0x1F moduli (31 in decimale) avremo ben 31 numeri che indicano ogni volta il numero di file presenti nei rispettivi moduli.

Questa sezione della fat la chiamiamo (i nomi che do sono puramente indicativi, ognuno li può chiamare come vuole!) **DimensionModuleList**.

La **FileList**:

00152efch: 62 6F 6F 74 2E 61 70 70 00 00 00 00 00 00 00 00 ; boot.app.....
00152f0ch: 00 00 00 23 62 6F 6F 74 76 61 72 2E 63 6C 61 00 ; ...#bootvar.cla.
00152f1ch: 00 00 00 23 00 00 00 61 63 61 5F 69 6E 69 74 2E ; ...#...aca_init.
00152f2ch: 63 70 69 00 00 00 00 84 00 00 01 13 64 69 73 70 ; cpi....„....disp
00152f3ch: 2E 63 6C 61 00 00 00 00 00 00 01 97 00 00 00 55 ; .cla.....—...U
00152f4ch: 64 69 73 70 6C 61 79 2E 63 70 69 00 00 00 01 EC ; display.cpi....ì
00152f5ch: 00 00 00 7F 6C 61 6E 63 65 2E 63 70 69 00 00 00 ; ... lance.cpi...
00152f6ch: 00 00 02 6B 00 00 00 1F 6C 69 6E 6B 62 6F 6F 74 ; ...k....linkboot
00152f7ch: 2E 63 70 69 00 00 02 8A 00 00 02 AC 6D 61 69 6E ; .cpi...Š...¬main
00152f8ch: 62 6F 6F 74 2E 63 70 69 00 00 05 36 00 00 01 38 ; boot.cpi...6...8
00152f9ch: 6D 65 6D 5F 7A 64 61 2E 63 70 69 00 00 00 06 6E ; mem_zda.cpi....n
00152fach: 00 00 00 B2 61 6C 2E 63 6C 61 00 00 00 00 00 00 ; ...²al.cla.....
00152fbch: 00 00 00 00 00 00 00 56 61 6C 61 72 6D 2E 61 70 ; .....Valarm.ap
00152fcch: 70 00 00 00 00 00 00 00 56 00 00 00 37 61 6C 61 72 ; p.....V...7alar
00152fdch: 6D 2E 63 6C 61 00 00 00 00 00 00 8D 00 00 00 79 ; m.cla..... ...y
00152fech: 61 6C 5F 66 69 6E 2E 63 6C 61 00 00 00 00 01 06 ; al_fin.cla.....
00152ffch: 00 00 00 4A 61 6C 5F 66 69 6E 2E 63 70 69 00 00 ; ...Jal_fin.cpi..
0015300ch: 00 00 01 50 00 00 01 F8 61 6C 5F 69 6E 69 74 2E ; ...P...øal_init.
0015301ch: 63 70 69 00 ; cpi.

Questa è un pezzo della FileList, ossia l'insieme di record che indica (indovinate un pò!) il nome del file e l'offset dello stesso rispetto all'inizio dei dati del modulo in cui è contenuto.
Come fare a capire se il file è di un modulo o di un altro? Beh ci sono due metodi, il primo è partire

a contare il numero di record e quando si è raggiunto il numero di file descritto nella dimensionmodulelist dopo incontreremo il primo file contenuto nel secondo modulo e via facendo! Più semplice di cosi...
Il secondo metodo è andare a controllare dove inizia la filelist del modulo preso in considerazione attraverso la tabella che si chiama AddressFileList e che andremo ad analizzare successivamente.

Analizziamo ora questa benedetta FileList:
I record in questo caso sono di 20 bytes, ma può capitare che siano anche di 32 o altro valore, comunque non cambia mai di molto la struttura, ossia c'è sempre il nome più l'offset del file.
Ecco, partiamo dal nome. Il nome rappresenta appunto il nome del file all'interno della struttura MHW e ha una lunghezza massima di 8 caratteri che sommati ai quattro dell'estensione diventano 12 caratteri!
Subito dopo si trovano 4 bytes che sono l'offset del file espresso in esadecimale.
Ma questo offset rispetto a cosa è considerato?
Vi ricordate la modulelist? No? beh era costituita dal nome più l'indirizzo dei dati dei file che conteneva... ecco... rispetto a questo indirizzo.

Esempio:
modulo boot -> indirizzo 0x2C11078C
Andiamo nella FileList, primo file boot.app -> offset 0x00000000
Prendiamo il secondo file bootvar.cla -> offset 0x00000023

Quindi per trovare l'indirizzo effettivo di boot.app dovremo fare 0x2C11078C + 0x00000000, invece per il bootvar.cla 0x2C11078C + 0x00000023, e via così...

Dopo i 4 bytes che indicano l'offset si trovano altri 4 bytes (da leggere in esadecimale) che indicano la lunghezza in bytes del file stesso. (in poche parole il file è costituito da questo numero di byte, i quali si andranno a leggere all'indirizzo base del modulo sommato dell'offset indicato)

Nel caso del file boot.app è appunto 0x23

ACTUNG, ACTUNG... o meglio attenzione! (non so nemmeno se lo ho scritto nel modo giusto ma mi piace la pronuncia)
Capita in alcune fat che la file list non sia subito dopo la dimensionmodulelist, e questo ci porta ad analizzare la addressfilelist!

La **AddressFileList**

001565e8h: 2C 17 2E FC 2C 17 2F B0 2C 17 33 D4 2C 17 35 B4 ; ,..ü,./°,.3Ô,.5´
001565f8h: 2C 17 36 68 2C 17 43 38 2C 17 43 B0 2C 17 44 8C ; ,.6h,.C8,.C°,.DŒ
00156608h: 2C 17 46 1C 2C 17 48 4C 2C 17 4A 7C 2C 17 4B 08 ; ,.F.,.HL,.J|,.K.
00156618h: 2C 17 4B 6C 2C 17 4B E4 2C 17 4D 9C 2C 17 4F 7C ; ,.Kl,.Kä,.Mœ,.O|
00156628h: 2C 17 51 98 2C 17 53 14 2C 17 53 F0 2C 17 55 1C ; ,.Q˜,.S.,.Sð,.U.
00156638h: 2C 17 56 84 2C 17 57 88 2C 17 58 A0 2C 17 59 04 ; ,.V„,.Wˆ,.X ,.Y.
00156648h: 2C 17 5E 68 2C 17 5E F4 2C 17 60 FC 2C 17 61 9C ; ,.^h,.^ô,.`ü,.aœ
00156658h: 2C 17 62 8C 2C 17 64 1C 2C 17 65 48 ; ,.bŒ,.d.,.eH

Finalmente... siamo giunti all'ultima parte di questa fat, ma vediamo che cosa fa questa sezione.

Niente di particolare, semplicemente indica per ogni rispettivo modulo a quale indirizzo si trova la FileList del modulo stesso.
Per ogni modulo ci sono 4 byte da leggere in forma esadecimale e ad ogni indirizzo va tolto l'address base (0x2C020000 nel nostro caso) per conoscere il reale indirizzo nel file.
Ovviamente e come sempre si considera il primo indirizzo per il primo modulo, il secondo indirizzo per il secondo modulo, ecc...

Capiamo perchè è importante questa sezione:
beh per prima cosa ci da un riferimento diretto alla posizione della FileList del modulo che cerchiamo, poi non è sempre detto che la FileList sia situata subito dopo la DimensionModuleList, mentre è assicurato che la AddressFileList sia vicina alla DimensionModuleList...
Inoltre la AddressFileList potrebbe essere separata in più spezzoni, cioè ogni modulo potrebbe avere la FileList posta in locazioni completamente differenti nel firmware. Se avete dato uno sguardo agli ultimi firmware originali che ci hanno messo a disposizione philips e strong (quelli su base nds), possiamo notare come questo avvenga. Se analizziamo correttamente il firmware vedremo che le filelist di ogni modulo sono situate ogni volta sotto la fine della zona dati del modulo stesso (quella che contiene i byte che costituiscono i file)

Con queste semplici e basiche informazioni è possibile estrarre tutto il MHW che costituisce un firmware, per l'occorrenza era stato fatto anche un semplice programma chiamato gluedump che permettava appunto di estrarre (fare il dump) di tutto il mhw e di ricompattare il firmware.

Facciamo attenzione a un piccolo particolare, compattare/decompattare è diverso da comprimere/decomprimere. Con compattare/decompattare intendo l'azione che si compie nell'estrarre e nel reinserire il MHW (i file/moduli) del firmware, mentre con comprimere/decomprimere intendo l'azione di passare un file per un algoritmo che provvederà a diminuire/'riportare al numero originale' le dimensioni di un file (praticamente come eseguire l'azione di zip o di rar sui file).

PS: è necessario che la ModuleList e tutte le altre sezioni (a parte la filelist) siano sempre poste allo stesso indirizzo originale, perchè all'interno del firmware sono presenti dei puntatori a queste zone. Nel caso sia necessario, si potrebbero modificare questi puntatori, ma non è sempre facile sapere dove stanno. Nel Nokia 9701s si conoscono, nel philips e nel pioneer questa cosa era stata fatta da dynamit e successivamente da molti altri personaggi, ma "l'indirizzo puntante" non è facile da trovare e impostare perchè fa appunto riferimento alla ram e bisogna stare attenti a come viene gestita, mentre nel nokia il tutto punta alla flash e successivamente viene caricato in ram quindi non ci sono problemi...

Dimenticavo.... parlo sempre di philips, pioneer, strong e nokia 9701s perchè questi sono i 4 grandi gruppi che si diversificano un pò nella struttura mhw in ram (non nel firmware). Da precisare che pioneer e philips però sono molto simili, i quali pare abbiano entrambi lo stesso sistema operativo, chiamato OS20 o qualcosa del genere!
Tutti gli altri decoder sono assimilabili a questi grandi gruppi, a volte bastano poche modifiche al codice e si riesce a estrarre qualsiasi fat, almeno per ora io non ho mai incontrato alcun problema nell'estrarre le varie fat.

I File MHW: come sono strutturati

Come abbiamo visto dentro un firmware ci sono più file compattati.
I file a seconda di che tipo sono, vengono compilati per poter essere compresi dal decoder.
A livello MHW si possono riconoscere diversi file che costituiscono le varie applicazioni.
Questi file sono contenuti in directory che rappresentano i vari moduli che vanno a formare un firmware.
Vediamo quali sono e a cosa servono principalmente:

- File di tipo "*.app" (applicazioni): Sono file che descrivono come deve comportarsi il decoder per poter eseguire l'applicazione. Al loro interno sono contenute le definizioni delle classi, i metodi di inizializzazione e di terminazione e il nome della color palette. A volte possono avere estensione "*.apz"
- File di tipo "*.cla" (classi): File che contengono le definizioni delle variabili usate negli script. A volte possono avere estensione "*.clz"
- File di tipo "*.cpi" (script): Sono i file che contengono le istruzioni da eseguire scritte in Pantalk. Si possono trovare a volte con estensione "*.cpz"
- File di tipo "*.lut" (color palette): In questi file sono contenuti i valori dei 16 possibili colori RGB e il loro livello di trasparenza. Alcune volte possono avere estensione "*.luz"
- File di tipo "*.ima" (immagini): Sono file che contengono le immagini usate dal firmware, ma contengono solo il numero del colore definito nella color palette usato per ogni pixel. Può capitare che questi file abbiano estensione "*.imz"
- File di tipo "*.ico" (librerie di icone): Questi file contengono un insieme di immagini, le icone per la precisione (piccole immagini). Capitano casi in cui questi file hanno estensione "*.icz"
- File di tipo "*.pan" (pannelli): File che contengono la descrizione di tutti gli elementi del decoder che vengono rappresentati sullo schermo (per esempio: i menu, le bande sullo schermo durante il cambio canale, ecc). Questi file possono anche avere estensione "*.paz" o "*.paa"
- File di tipo "*.dat" (dati): File al cui interno sono presenti dati di configurazione, ecc. Possono avere anche estensioni "*.daz"
- File di tipo "*.mpg" (stringhe di testo): Sono file che contengono stringhe di testo. A volte hanno estensione "*.mpz"
- File di tipo "*.bin" (binari): File che contengono dati che vengono sfruttati dal firmware
- File di tipo "*.fnu" (file mpeg): File che contengono delle immagini compresse con il noto algoritmo di compressione mpeg-2. Per poterli utilizzare normalmente, bisogna rinominarli con estensione *.m2v. Questo tipo di file, a differenza degli altri, è l'unico che non deve essere compresso durante l'inserimento nel firmware.
- File senza estensione: Vengono usati dagli script per avere alcuni tipi di informazione.

Da qui in avanti descriverò la struttura di tutti i file MHW, mi ha portato via un sacco di tempo capire come sono strutturati e solo capendoli sono riuscito a far funzionare in modo perfetto defiant con la gestione di questi file, vorrei per questo che tutto il tempo che ho perso io non sia andato invano e che magari possa servire ad altri...

Vediamo in dettaglio i file con estensione ***.app** come sono strutturati (ovviamente si fa riferimento alla struttura una volta che il file è stato decompresso)

Indirizzo	N° bytes	Descrizione

0x00	2 Bytes	Sono sempre a 0x00
0x02	Y Bytes	Nome script iniziale
0xYX	W Bytes	Nome script finale
0xYY	H bytes	Nome Default Palette
0xHZ	2 Bytes	N° di classi
0xHZ+2	J Bytes * N° di classi	Nomi delle classi
0xZZ	10 Bytes	Sempre a 0x00
0xZ	2 Bytes	N° di commenti
0xZ+2	F Bytes * N° di commenti	Testo dei commenti
0xHV	38 Bytes	Sempre a 0x00

La struttura dei record per i nomi e i commenti è uguale a quella per le stringhe di testo nei file di testo (*.mpg), ossia due byte iniziali che indicano il n° di caratteri della stringa e consecutivamente i caratteri che formano la stringa stessa (vedremo meglio successivamente).

I file con estensione ***.cla**

Indirizzo	N° bytes	Descrizione
0x00	2 Bytes	N° di variabili
0x02	4 Bytes	Sempre a 0x01 0x01 0x00 0x00
0x06	2 Bytes	N° di bytes per allocazione della memoria in base alla tabella sotto-riportata
0x08	2 Bytes	N° Byte per la descrizione delle variabili
0x0A	2 Bytes	Lunghezza del nome della classe + 1
0x0C	2 Bytes	Lunghezza dei nomi delle variabili
0x0E	X Bytes	Descrizione variabili in base alla tabella sotto-riportata
0xZZ	Nome classe + Nomi variabili	Ogni nome è diviso da un byte a 0x00
0xFine-2	2 Bytes	Sempre a 0x00

Tabella Descrizione Variabili:
Le variabili vengono gestite nel seguente modo:
- Se il 1° bit (il Byte AND 0x80) è a 1 allora è presente la seconda dimensione, altrimenti no
- I bit dal 5° al 8° (il Byte AND 0x0F) indicano il tipo di variabile in base alla tabella riportata sotto
- Il 2° byte viene gestito nel seguente modo:
- Se è una variabile Intera allora rappresenta il numero di digits

- Se è una variabile Reale allora rappresenta dal 1° bit al 4° bit la precisione (il Byte AND 0xF0), e dal 5° al 8° la parte intera (byte AND 0x0F)
- Se è una variabile di tipo Testo o Memory o Alpha o Alphanum allora rappresenta il minuscolo (se a 1 allora la variabile conterrà caratteri sia minuscoli che maiuscoli, se invece è a 0 conterra solo caratteri Maiuscoli). In questo caso sono presenti, dopo tale byte, altri 2 byte che indicano il N° di caratteri-byte della variabile.
- Il 3° byte (5° nel caso testo o memory) è presente solo se la variabile è a due dimensioni:
- 2 byte indicano la 1° dimensione
- 2 byte indicano la 2° dimensione

Tabella Tipi di Variabili:

Valore	Tipo
00	ALPHA
01	ALPHANU
02	INTEGER
03	REAL
06	TIME/FORMAT
08	TEXT
09	MEMORY

Tabella Allocazione Memoria:

Tipo	N° Bytes da riservare
REAL	8 Bytes
INTEGER	4 Bytes
ALPHA	((N° caratteri-byte) AND 0xFFFE) + 2
ALPHANU	4 + N° caratteri-byte
MEMORY	((N° caratteri-byte) AND 0xFFFE) + 2
TIME/FORMAT	((N° caratteri-byte) AND 0xFFFE) + 2
TEXT	((N° caratteri-byte) AND 0xFFFE) + 2

Ogni parziale di ogni variabile che viene calcolato su ogni variabile deve essere arrotondato per eccesso a MOD 4 (Multiplo di 4) e poi sommato al totale.
Quando sono presenti una o più dimensioni (array a 1 o due dimensioni) bisogna calcolare come sopra il valore dell'allocazione per la variabile, ma prima di arrotondarla e sommarla al totale, bisogna moltiplicare il valore parziale trovato per la prima dimensione, poi per la seconda e sommare 4. Una volta fatto ciò si procede a fare l'arrotondamento del valore per 4 e a sommarlo al totale.
Alla fine, quando si è trovato il N° di byte da allocare, il valore totale deve essere arrotondato per eccesso a MOD 8 (Multiplo di 8).

I file con estensione ***.cpi**

Indirizzo	N° bytes	Descrizione
0x00	2 Bytes	Lunghezza in byte della zona riguardante il codice in OpCode (Z bytes)
0x02	16 Bytes	Default Class
0x12	1 Byte	N° di classi che utilizza lo script
0x13	(16 Bytes + 2 Bytes) * N° di classi	Nome della classe + due byte che indicano uno il terminatore uguale a 0x00 e l'altro come è dichiarata la classe (private=108 o public=0)
0xZZ	Z Bytes	Codice in OpCode dello script

Per sapere come decompilare gli script nella parte riguardante l'opcode vi rimando al paragrafo L'OpCode.

I file con estensione ***.lut**

Indirizzo	N° bytes	Descrizione
0x00	4 Bytes * 16 colori	Descrizione dei colori
	3 Bytes	Colori RGB
	1 Byte	Trasparenza

Se la trasparenza è uguale a:
- 0 allora = “Trasparente 100%”
- 1 allora = “Opaco”
- 2 allora = “Opaco 75%”
- 3 allora = “Opaco 50%”
- 4 allora = “Opaco 25%”
- 5 allora = “Opaco 10%”

I file con estensione ***.ima**

Indirizzo	N° bytes	Descrizione
0x00	2 Bytes	Larghezza
0x02	2 Bytes	Altezza

0x04	Altezza*(Larghezza/2) Bytes	N° del colore in Palette

Le immagini possono avere qualsiasi dimensione. Il numero massimo di colori deve essere di 16.

I file con estensione ***.ico**

Indirizzo	N° bytes	Descrizione
0x00	2 Bytes	Lunghezza totale della libreria – 4 bytes
0x02	2 Bytes	N° di icone
0x04	2 Bytes * N° di icone	Offset della posizione della descrizione dell'icona
0xQ	X Bytes	Nome dell'Icona, è terminato dal byte 0x00
0xQ+len(Nomeicona)	1 Byte	Larghezza icona divisa per 2
0xQ+len(Nomeicona)+1	1 Byte	Altezza icona
0xQ+len(Nomeicona)+2	1 Byte	Byte divisore a 0x00
0xQ+len(Nomeicona)+3	Altezza*(Larghezza/2) Bytes	Descrizione dell'icona

La tabella usata per descrivere le icone non è altro che il numero in successione del colore relativo al pixel presente nel file *.lut (palette). Per indicare il numero del colore si utilizzano solo 4 bit (un Nibble, un Byte descrive due pixel). La dimensione delle varie icone presenti nel file devono avere larghezza multipla di 4.
Il numero massimo di colori deve essere di 16 e le dimensioni devono essere inferiori a 256*256.

I file con estensione ***.pan**

Indirizzo	N° bytes	Descrizione
0x00	2 Bytes	Metodo di inizializzazione
0x02	2 Bytes	Metodo di aggiornamento
0x04	2 Bytes	Metodo di finalizzazione
0x06	2 Bytes	Maschera della lingua (bit da 4 a 0, se bit è a 1 allora lingua selezionata. Lingua 0 = stato bit 0, ecc)
0x08	1 Byte	Lingua predefinita
0x09	U Bytes	Nome Classe
	Y Bytes	Nome libreria di icone
	2 Bytes	Lunghezza totale dei nomi (Nome pannello, nomi delle variabili, nome next panel, nomi degli elementi, per ogni

		elemento di tipo 3 va aggiunto 0x20)
	2 Bytes	Lunghezza totale delle stringhe della lingua 0 (ogni lunghezza di stringa prima di essere sommata al totale viene arrotondata a Word)
	2 Bytes	Lunghezza totale delle stringhe della lingua 1 (ogni lunghezza di stringa prima di essere sommata al totale viene arrotondata a Word)
	2 Bytes	Lunghezza totale delle stringhe della lingua 2 (ogni lunghezza di stringa prima di essere sommata al totale viene arrotondata a Word)
	2 Bytes	Lunghezza totale delle stringhe della lingua 3 (ogni lunghezza di stringa prima di essere sommata al totale viene arrotondata a Word)
	2 Bytes	Lunghezza totale delle stringhe della lingua 4 (ogni lunghezza di stringa prima di essere sommata al totale viene arrotondata a Word)
	2 Bytes	N° di widgets
Per ogni Widget la seguente tabella		
	1 Byte	Tipo di widget in base a tabella sotto-riportata
	1 Byte	2 bit per ogni valore dei quattro byte successvi (i due bit sono rispettivamente i bit 9 e 8 del valore del byte dopo. In ordine troviamo bit 7-6 Altezza, bit 5-4 Larghezza, bit 3-2 Coordinata Y, bit 1-0 Coordinata X)
	1 Byte	Coordinata X (da aggiungere ai due bit sopra riportati)
	1 Byte	Coordinata Y (da aggiungere ai due bit sopra riportati)

	1 Byte	Larghezza (da aggiungere ai due bit sopra riportati)
	1 Byte	Altezza (da aggiungere ai due bit sopra riportati)
Esempio: bit 0-1 concatenato ai bit del byte coordinata X = bit da 9-0 della coordinata X, gli altri bit (15-10) sono=0. X min=0, Y min=0, X max=576, Y max=486. Le cordinate ricavate vanno sommate a quelle del PanBox		
	1 Byte	Sempre 0x00
	1 Byte	Gruppo del widget
	1 Byte	Prima pagina nella quale appare il widget
	1 Byte	Priorità dell'elemento
	1 Byte	Stato dell'elemento in base a tabella sotto-riportata
	1 Byte	Posizione
	1 Byte	Font usato
	1 Byte	Giustificazione (bit 0-1) (se 0 allora sinistra, 1 centro, 2 destra) e dimensione Font usato(bit 2-7)
	1 Byte	N° di stili
Per ogni stile		
	1 Byte	4 bit colore sfondo e 4 bit colore bordo
	1 Byte	4 bit colore testo e 4 bit stile tipo bordo (in base a tabella sotto riportata)
	1 Byte	4 bit larghezza bordo1 e 4 bit larghezza bordo2
Fine Stili		
	4 Bytes	Page bitmap, se bit 0 attivo allora l'elemento è visualizzato su schermo 0, ecc (max 32 schermi). Se il tipo widget=1 allora il campo vale 1.

	2 Bytes	Reserve Code, il codice che fa eseguire il metodo associato(in base a tabella sotto-riportata)
	2 Bytes	Indice del vettore della variabile
	2 Bytes 2 Bytes	Sempre 0x0000 Metodo dell'elemento
	Z Bytes	Nome del widget
	R Bytes	Nome variabile
	W Bytes	Nome Panel Successivo
Se il tipo widget è diverso da 3 (InputBox)		
	P Bytes	Testo dell'elemento
Se status byte AND 4 = 4 (Multi-Lingua)		
Per ogni lingua abilitata, esclusa la lingua 0		
	P Bytes	Testo dell'elemento nella lingua solo se abilitata
Fine Lingue abilitate		
Fine Se status byte		
Fine Se tipo widget		
	1 Byte X Byte	N° di font Nomi dei font

I nomi vengono interpretati nel seguente modo:
il primo byte rappresenta la lunghezza della stringa, di seguito troviamo i caratteri che la compongono.
Solo nel caso del "testo dell'elemento", i byte che indicano la lunghezza della stringa sono due.
Gli elementi vengono disegnati in ordine sull'asse delle Z in base alla loro posizione nel file del pannello, l'ultimo widget descritto dal file è quello che sta davanti a tutti gli altri widget del pannello.
Il primo elemento è sempre il PanBox.

Tabella Tipi di Widget

Valore	Tipo
1	PanBox
2	Button
3	InputBox

4	TextBox
5	Frame
6	SPanel
7	ListBox
8	EnumBox
9	Counter
10	SubPanel
11	CheckBox
12	RadioBox
13	BarGraph
14	IconBox
15	EditBox
16	Window
17	Slider
18	Panel
19	PoList

Tabella stile tipo Bordo

Valore	Tipo
0	Linea
1	Linea Arrotondata
2	Smussatura in rilievo
3	Smussatura incassata
4	Composito 1
5	Composito 2

Tabella Reserve Code

Valore	Tipo
0	Nessuna chiave associata
0x259	0
0x25A	1
0x25B	2
0x25C	3
0x25D	4

0x25E	5
0x25F	6
0x260	7
0x261	8
0x262	9
0x263	A
0x264	B
0x265	C
0x266	D
0x267	E
0x268	Pilota
0x269	Guida
0x26A	X
0x26B	o->
0x26C	Pers
0x26D	+
0x26E	Serv
0x26F	TV/Sat
0x270	Mute
0x271	StandBy
0x272	UP
0x273	DOWN
0x274	LEFT
0x275	RIGHT
0x276	OK

Tabella Byte di Stato

Bit	Descrizione
0	Abilitato
1	Sconosciuto
2	Multi-Lingua
3	Esci dal pannello
4	Bitmap
5	Nascosto

6	Non disegnare
7	Sempre zero

I file con estensione ***.mpg**

Indirizzo	N° bytes	Descrizione
0x00	2 Bytes * Ogni testo nel file	Indirizzo di inizio dei dati del testo (una coppia di byte per ogni testo)
0xWW	X Bytes	Il primo byte indica la lunghezza del testo (se è a stringa lunga i primi due byte indicano la lunghezza), poi sono presenti i caratteri che compongono la stringa

Ci sono due tipi di file quelli a stringa lunga e quelli a stringa corta. I file a stringa lunga sfruttano due bytes per indicare quanto è lunga la stringa interessata. I file a stringa corta usano un solo byte per indicare quanto è lunga la stringa relativa.

I file con estensione ***.dat, *.bin e i file senza estensione*** vanno letti in modo binario, la loro struttura non è uno standard definito in quanto potrebbero essere semplici file oppure file con strutture definite per essere letti via MHW.

I file con estensione ***.fnu** essendo file compressi in mpeg-2 vanno considerati per quello che sono, ossia file mpeg-2 e epr questo vanno interpretati in base a questo standard che è disponibile su internet e definito come standard internazionale.

Fonts: Fudemi, Videotex

Questa volta parliamo dei font.
Che sono? I font sono l'insieme delle descrizioni dei caratteri.
Ogni font presenta una serie di caratteri che possono essere più o meno uguali tra di loro.
L'importante è che ci sia la descrizione dei caratteri che poi verranno visualizzati quando si avrà un output di tipo testuale usando il MHW.

I font sono un argomento un pò delicato perchè sebbene facciano parte del MHW non sono sempre estraibili dal firmware, a volte sono addirittura non presenti (o se ci sono non hanno la tipica struttura di font MHW)

Anche qui le tabelle variano a seconda della tipologia di firmware usato, ma ci sono principalmente due grandi categorie, quelle dei pioneer e quella degli strong.
Quindi è molto probabile che il vostro firmware possegga una struttura simile a una di queste due grandi categorie.

Questo paragrafo inizia come una bozza, forse riuscirò a cercare di esplicarlo meglio, ma per ora voglio che ci sia almeno qualche appunto.
Descriviamo per prima la struttura dei font degli **strong**.

Precisamente andremo a descrivere i font dei **Nokia (9701s/9850) e Sony**

Ogni fontset descrive 256 caratteri, di questi i primi 32 sono caratteri vuoti!
I successivi 128 caratteri rappresentano sempre (più o meno) i 128 caratteri standard della tabella ASCII.
Gli altri sono definibili a seconda dei casi, ma tra i vari firmware rappresentano sempre le stesse figure (anche qui c'è un'approssimazione abbondante perchè capita che in alcuni firmware è una freccia in altri è una faccina).

Nel firmware è presente un puntatore a una tabella che chiamiamo **AddressFontList**

La AddressFontList è formata da 4 bytes per ogni carattere di ogni fontset. Ossia avremo (256*4)*NumberFontSet bytes. Indica l'indirizzo della FontList del rispettivo carattere.

Finita la AddressFontList troviamo il **NumberFontSet** è rappresentato da 4 bytes (in esadecimale) e che indica il numero di fontset che il firmware contiene.

Dopo il NumberFontSet abbiamo la **AddressFontInfo** dei record formati da 4 bytes che rappresentano l'indirizzo della tabella FontInfo del fontset rispettivo.

Successivamente troviamo la **AddressFontTable** formata da record di 4 bytes che indicano l'indirizzo della fonttable del rispettivo fontset.

Andiamo ora ad analizzare la **FontInfo**:
questa è costituita da 13 bytes, i primi 8 sono il nome del fontset, 1 byte che indica la larghezza del fontset predefinita, un byte che indica l'altezza in pixel del fontset (se vale 0 è da considerarsi come se valesse 12), un byte che indica la dimensione del fontset, un byte che indica il FontSetId (leggibile dalla funzione MHW Get_Font_ID(). Sembra avere lo stesso significato del font name, tutti i font con lo stesso nome hanno lo stesso FontID), un bytes il cui significato è sconosciuto al momento.

La **FontTable** è costituita da un byte per ogni carattere che rappresenta la larghezza in pixel del carattere corrispondente.

La **FontList** è costituita dall'insieme dei bits che vanno a formare i caratteri del fontset. Quindi avremo AltezzaFontSet*LarghezzaPixelCarattere bits che costituiranno il nostro carattere. Questo valore va prima convertito in byte e arrotondato affinchè non ci siano mezzi byte (arrotondamento a byte singolo), successivamente a Word (4 bytes). Il numero di byte ottenuti saranno il numero di bytes che andranno a descrivere il nostro carattere nella FontList.
Dei bytes letti dovremo associare ogni bit ad ogni pixel che forma il carattere, di conseguenza se il bit vale 0 il rispettivo pixel è spento (non colorato), altrimenti (se vale 1) sarà acceso/colorato.

Per i firmware di **Strong e Asscom** abbiamo un'unica differenza:

La AddressFontList è costituita da 4*3*NumberFontSet bytes. (non conosco al momento il significato di questi bytes ma sospetto siano l'indirizzo delle fontlist dei primi tre caratteri, non ho mai verificato).

Altra piccola cosa... il font 'fudemi' deve sempre esistere nei firmware, spesso si trovano tre fontset (due 'fudemi', un 'videotex' inutilizzato).
Nel Nokia 9850 si ha il font 'eurost' al posto del 'fudemi'.

Si è provato recentemente a sostituire il font videotex con fudemi e si è riuscito con successo, l'unico problema è che come al solito bisogna fare anche qualche piccola modifica ai puntatori in HDL.

Ultimo appunto: le tabelle dei vari address (AddressFontInfo, AddressFontList, AddressFontTable, NumberFontSet) sono sempre unite tra di loro, spesso (per non dire sempre) anche quelle che costituiscono i vari FontSet (FontTable, FontInfo, FontList) ma non è necessario visto che ci sono gli indirizzamenti a queste zone (appunto le varie table address).

Per i firmware con base **pioneer/philips** i font hanno la stessa struttura dei Nokia (9701s/9850) e Sony. C'è solo una piccola differenza, i vari indirizzi vanno letti in little-endian.

Per trovare i font nel firmware basta cercare la parola 'fudemi', individuare a quale offset del firmware si trova e aggiungere il valore dell'indirizzo base del firmware in memoria (ram o flash a seconda dei casi). Una volta ottenuto l'indirizzo effettivo basta cercare questo valore all'interno del firmware e ci troveremo la posizione delle tabelle degli address dei font, in questo modo potremo accedere a tutte le altre tabelle.

Con defiant è possibile modificare i font per molti firmware (ma non tutti, solo quelli che hanno le strutture dei font conosciute!).

Per editare i font bisogna cliccare su modifica-> editor font, oppure il pulsante nella barra che si chiama modifica font...

Vi si aprirà una finestra dove potrete impostare tutte le opzioni per modificare i font, selezionando un font potrete decidere quali pixel colorare e quali no. Mi sembra molto semplice da utilizzare quindi non mi dilungherò molto.

Volevo solo farvi notare che è possibile importare ed esportare i singoli caratteri! Sembra una sciocchezza ma è davvero molto utile!

E' anche possibile importare o esportare tutti i font interi con un semplice click su inserisci lista dei font o esporta lista dei font... sono i pulsanti di fianco a quello per l'editor dei font nella barra.

Nel caso il vostro firmware non sia supportato per l'estrazione dei font è possibile utilizzare dei font esterni attraverso dei file. Questi file sono già presenti nella cartella principale di defiant tanto per farvi vedere un esempio e hanno estensione *.font.
E' possibile scegliere di utilizzare i font esterni tramite menu configurazione-> font-> usa file esterno.
L'opzione "tabelle delle lunghezze all'inizio" serve per cercare le informazioni sui font nel file esterno all'inizio del file (insomma serve per la struttura del file dei font!)

La struttura HDL

Le funzioni HDL come detto in precedenza non sono altro che le funzioni che gestiscono le periferiche del nostro STB.
La particolarità di queste funzioni HDL è che sono state compilate in linguaggio macchina, a differenza del pantalk che è interpretato dalla Virtual Machine.

Le funzioni HDL, tanto per darvi un esempio, sono le varie chiamate alla routine MHW UndeclareEvent, Tcs_Next_Channel, ecc...

Quando in pantalk noi scriviamo uno script ovviamente utilizziamo funzioni base che ci offre il linguaggio. Oltre alle funzioni base ci sono quelle che generalmente vengono chiamate funzioni HDL, ossia le funzioni che sono stabilite/definite in base al tipo di funzione per cui deve essere eseguito il pantalk.
Ad esempio sui decoder c'è la chiamata device_io che viene utilizzata per impostare le varie opzioni delle device che presenta un STB.

Spesso sentiamo parlare di modifiche HDL, ma perchè?
Beh semplicemente perchè a volte è necessario interagire direttamente con l'hardware e applicare nuove opzioni alle funzioni già esistenti.

Come fare per modificare una funzione HDL? Beh... l'unica è cercare di capire come funziona una funzione HDL tramite la sua disassemblazione in codice asm. Una volta capito cosa fa l'asm si modifica a piacimento la funzione in base alle proprie esigenze.
Non è necessario capire cosa faccia l'asm, si potrebbe riscrivere da zero la funzione e andarla a sostituire direttamente, ma il problema principale è che spesso di tutto ciò che interagisce in HDL non si ha la ben che minima documentazione ed è necessario prima capire come funziona tutto quanto per poter poi effetturare le dovute modifiche.

Mhm... questo è in generale l'HDL.

Ora voi vi chiederete... come cavolo faccio io a sapere dove si trova una certa funzione all'interno del firmware? Come faccio a sapere che corrisponde a quel dato nome?
Beh c'è stato qualcuno (più di qualcuno) che ha unito le forze e hanno cercato di capire cosa facesse bene o male ogni funzione dandogli un nome più o meno significativo.
Le funzioni in questione sono parecchie e ognuna include altre sottofunzioni, però la maggior parte si è riuscite ad associarle ad alcune funzioni "standard" del pantalk e hanno permesso di far andare avanti lo sviluppo dei firmware
C'è una tabella, chiamata PDT, che permette di avere la certezza dell'associazione della funzione xxx alla posizione yyy nel firmware.

MHW: esecuzione/interpretazione del pantalk

Finalmente... dopo aver fatto una visita generale ai vari file possiamo ora cercare di comprendere come funziona realmente il MHW.

Come abbiamo visto un firmware MHW è composto da moduli. Questi moduli sono delle sorte di applicazioni nelle quali avvengono determinate azioni che deve compiere il firmware stesso. Praticamente è come suddividere il firmware in tante piccole parti.

Ogni modulo può essere allocato/deallocato in memoria, affinchè sia possibile accedervi o far eseguire del codice presente in esso.

Ma come facciamo a sapere da che punto partire? Beh sembra complesso ma è piuttosto intuitivo... esiste il modulo boot!
Questo modulo è sempre il primo ad essere eseguito ed è sempre il primo modulo in ordine di posizione che deve essere presente nella fat MHW.

Oltre al modulo boot ci sono altri due moduli che sono sempre presenti e che normalmente assumono sempre le solite funzioni in tutti i firmware (ho provato a creare firmware senza questi due moduli e funzionano perfettamente senza alcun problema, l'unico necessario è il boot!).
Questi due moduli sono il basic e il commun.
Il basic è il modulo considerato principale, ossia quello che normalmente si occupa dell'intercettazione degli eventi che accadono durante l'utilizzo normale del decoder (cambio canale, visione tv, accesso ai vari menu ecc...).
Il modulo commun è un modulo che possiamo considerare globale. Avete presente le variabili globali dei programmi? Ecco... cerchiamo di vedere il commun come la zona in cui andare a piazzare tutti i nostri dati di carattere generale e che ci possono servire anche negli altri moduli. Il commun viene caricato in memoria durante l'inizializzazione del firmware (inizializzazione MHW e non HDL... il boot invece viene caricato dalla parte HDL), assieme al modulo basic, e non viene mai deallocato fino allo spegnimento del decoder.

Ora abbiamo una buona base di partenza di conoscenze per poter cercare di cominciare, ma ci manca ancora qualcosa...
Abbiamo i nostri moduli, sappiamo che il primo ad essere eseguito è il boot... ma come cappero faccio a sapere che cosa guardare? Insomma... il modulo boot è diviso in diversi file dei quali ognuno compie un'azione, ma come faccio a sapere quale è il primo tra tutti quanti? Possibile che non ci sia una logica di avvio?

Infatti c'è! Entrano in gioco a questo punto i nostri file *.app, nei quali sono presenti parecchie informazioni su come avviene il caricamento in memoria del modulo.

Detto tutto questo uno potrebbe iniziare a lavorare tranquillamente sul MHW, ma mi piacerebbe entrare un pò più in dettaglio sul significato di ogni parametro di cui sono costituiti i vari tipi di file MHW.

Per questo scopo vi consiglio di procurarvi il Defiant nel caso non lo abbiate già fatto. (ovviamente l'ultima versione disponibile...)
Il Defiant è un programma tutto fare, dedicato alla modifica/interpretazione dei file MHW. E' un vero e proprio tool di sviluppo per il pantalk (io lo considero persino migliore di quella robaglia creata dalla ditta madre del pantalk, nel caso vi interessi potete cercare la versione demo del programma HPanel, che fantasia un programma che riprende il nome della ditta HyperPanel, la casa madre, o gratuita di PCBoot Direct, sempre della HyperPanel, al momento non credo siano più rintracciabili in internet, ma magari farò un paragrafo dove spiegherò come utilizzare anche questi "aggieggi" così capirete che rivoluzione è stato defiant!)

Come va utilizzato defiant:
è un programma molto intuitivo ma essendo specifico per il pantalk probabilmente non tutto è

sempre comprensibile a primo occhio.

Partiamo con l'apertura del firmware, clicchiamo su apri e dalla finestra di scelta del file ricordiamoci di selezionare il giusto tipo di firmware nell'apposita casella a cascata che c'è sotto il nome del file scelto. Il tipo di firmware normalmente è diviso in base al tipo di decoder a cui si fa riferimento.

Una volta aperto il firmware vi appariranno tutti moduli rappresentati come delle directory/cartelle dove sono presenti dei file. Questi file sono i file MHW che sono stati decompattati dalla FAT del firmware. Eh si visto come è semplice?

Per aprire un file basta fare doppio click sul file stesso oppure cliccando con il destro selezionare apri, oppure premere INVIO una volta che il file è stato selezionato.
Defiant aprirà a seconda del tipo di file scelto una finestra che permetterà di editare il file in base alle sue caratteristiche.

I file di applicazione (*.app) sono i file che vengono utilizzati per indicare al modulo cosa fare quando viene allocato/caricato in memoria.

Apritene uno e vediamo in dettaglio cosa vi presenta la finestra che vi viene mostrata:
- script iniziale: è il nome dello script (senza l'estensione) che viene eseguito quando il modulo viene caricato in memoria
- script finale: è il nome dello script (senza l'estensione) che viene eseguito quando il modulo viene deallocato dalla memoria
- palette predefinita: è il nome della palette (senza l'estensione) che viene utilizzata di default per ogni pannello, o file che necessita di palette, del modulo nel caso non se ne impostasse una via codice.
- commenti: sono dei commenti, ci si può scrivere quello che uno vuole, non conosco il significato e non influenzano l'esecuzione del firmware semplicemente ogni tanto se ne trovano nei firmware originali (commenti che indicano delle directory o sezioni)
- classi: attraverso questa impostazione è possibile aggiungere o togliere le classi del modulo che vengono allocate in memoria ram quando il modulo viene caricato. Se non si caricano le classi in ram non sarà possibile utilizzarle all'interno degli script che non sono presenti nel modulo (vedi classi dichiarate private). Anche qui i nomi delle classi sono indicati senza l'estensione del file.

Dimenticavo... i file di applicazione sono presenti uno per ogni modulo e devono avere lo stesso nome del modulo.

I file di classi (*.cla) contengono al loro interno tutte le dichiarazioni delle variabili che si utilizzano nei vari script del firmware.

Aprendo un file di questo tipo si aprirà una finestra che ci presenta un listato di nomi. Questi nomi sono i nomi delle nostre variabili. Selezionando una variabile è possibile vedere a destra tutte le caratteristiche che la compongono.

Ma vediamo meglio di che si tratta:
Classe -> Nome: è il nome della classe, è sempre uguale al nome del file e viene utilizzato per far

riferimento alla classe stessa, provando a cambiarlo non accade nulla (almeno sembra), ma è buona norma lasciare nome del file e nome reale della classe uguali (all'interno degli script si fa riferimento al nome della classe impostato in questo campo e non al nome del file)
Dichiarazione Variabile:
Nome: è il nome della variabile. Serve per poter fare riferimento alla variabile negli script. All'atto della compilazione degli script non si farà più riferimento al nome ma alla posizione della variabile all'interno della classe. E' necessario per questo che non si spostino le variabili dalla loro posizione altrimenti si incorrerà nel rischio di fare riferimento a variabili sbagliate (magari questo discorso lo riprenderemo quando parleremo degli script)
Tipo: Rappresenta il tipo di variabile. Abbiamo a disposizione una decina di tipi di variabile, quattro dei quali non si conosce il significato. (probabilmente non hanno alcun significato in quanto non sono mai utilizzati)

Qui bisogna aprire una parentesi sui tipi di variabile:
ALPHA: rappresenta le variabili che possono contenere dati di tipo carattere ma non numerici! Un singolo byte viene considerato un carattere, lunghezza di questo tipo variabile è 1 carattere.
ALPHANUM: rappresenta le variabili che possono contenere gli alphanumerici (caratteri e numeri). Stringhe lunghe fino a 255 caratteri.

Il parametro minuscole che appare con questi tipi di variabili serve per indicare se i caratteri al loro interno saranno solo maiuscoli o anche minuscoli!
Dimensione1 e Dimensione2 rappresentano il limite degli array o matrici. Nel caso si indichi dimensione1 maggiore di 1 si avrà un array, nel caso si indichi dimensione2 maggiore di 1 si avrà una matrice di quella dimensione.
E' come dichiarare in c la variabile vettore: vettore [10]
e la variabile matrice: matrice [10][5]
Il parametro lunghezza indica quanto è lunga la stringa massima che è possibile memorizzare nel tipo alpha o alphanum.

In sostanza abbiamo qualcosa di simile se fosse dichiarato in C:

```
typedef char alpha_alphanum[lunghezza];

alpha_alphanum matrice[dimensione1][dimensione2];
```

Il tipo di variabile INTEGER è un tipo intero di 4 byte quindi fate voi i vostri calcoli per sapere fin dove potete arrivare con il numero... io non me lo ricordo! Nel tipo intero troviamo un nuovo campo, il digits. Non so bene a cosa serva ma credo indichi quanti digit (singole cifre) può contenere. Normalmente questo valore è di 10 e io non lo ho mai variato.

Il tipo REAL è un dato reale. Anche qui troviamo due nuovi campi. Precisione e parte intera. Parte intera sono i digit della parte intera, precisione quelli della precisione

Il tipo FORMAT è un dato un pò particolare. Esistono funzioni MHW specifiche che lo sfruttano e diciamo che generalmente rappresenta dati formattati, come la data, l'ora e cose del genere. Ovviamente è una sorta di stringa quindi anche qui abbiamo la lunghezza come per gli alpha e alphanum che ci indica la lunghezza della stringa e poi abbiamo un nuovo campo: il gruppo.
Il gruppo è un dato che ci serve se utilizziamo certe funzioni MHW e va impostato all'occorrenza

per indicare il gruppo a cui appartiene la variabile. Infatti è possibile eseguire operazioni su queste variabili format in base al loro gruppo di appartenenza.
Questi quelli fin'ora incontrati:
00 (usato per numeri)
01 (usato per orario)
02 sconosciuto (forse per un uso generale)
0A (usato per input text)

Set_Input_Format($CURRENT, 0x0, "++.++", "DD.DD", &enr.ER);

il secondo parametro di questa funzione potrebbe essere
in qualche modo influito dal numero del gruppo... purtroppo c'è ancora qualche alone di mistero su questa funzione e sul suo utilizzo. Comunque si riesce a utilizzare lo stesso.

Il tipo TEXT è una stringa vera e propria e anche qui come per alpha e alphanum abbiamo le stesse caratteristiche. Possiamo avere delle stringhe lunghe fino a 64KB.

Il tipo MEMORY è un puntatore alla memoria presa in considerazione come singoli byte e come tale dobbiamo fare alcuni appunti.
Se impostiamo la lunghezza a 0 allora è un puntatore semplice, se invece impostiamo la lunghezza maggiore di 0 abbiamo un vettore di byte. Ovviamente dimensione 1 e 2 vengono utilizzati per creare matrici a più dimensioni... come avviene per tutti gli altri dati!

Il pulsante “Mostra/Nascondi visualizzazione testuale” non fa altro che visualizzare in modo testuale la dichiarazione delle variabili all'interno della classe. Questa funzione non ha alcun effetto sulla modifica della classe, serve solo per avere una visuale complessiva migliore della classe.

Mettendo il segno di spunta sull'opzione “Visualizza indice delle variabili” e cliccando su esegui verranno visualizzati gli indici delle variabili a fianco delle dichiarazioni delle stesse.

Mettendo il segno di spunta sull'opzione “Ordina variabili per nome” e cliccando su esegui verranno visulizzate le dichiarazioni delle variabili in ordine alfabetico per nome.

Mettendo un testo nella casella “Ricerca” potrete cercare del testo all'interno della visualizzazione testuale delle variabili. L'opzione “solo parole complete”... inutile che ve la spieghi perchè è una funzione comune alle ricerche di tutti i programmi :p

Cliccando sul pulsante “Trova variabili doppie” verranno mostrate tutte le variabili che sono state dichiarate con lo stesso nome all'interno della classe

Cliccando su “nuova variabile” verrà aggiunta una variabile alla posizione selezionata nella lista delle variabili.

Cliccando su “cancella variabile” verrà cancellata la variabile selezionata nella lista delle variabili.

I file di palette (*.lut) sono i file che contengono una tabella dei colori che può utilizzare un pannello per visualizzare i suoi "oggetti" (questi oggetti si chiamano widget) sullo schermo.

Apriamo un file di questo tipo con defiant e vi si aprirà una finestra con 16 quadrati colorati!

Cliccando col pulsante destro del mouse sul colore potremo cambiare facilmente il colore stesso grazie alla tabella standard dei colori di windows.
Cliccando sul colore col pulsante sinistro apparirà un menu nel quale sarà possibile scegliere il livello di trasparenza del colore. Il livello di trasparenza indica quanto il colore lascierà intravedere l'immagine che c'è sotto (spesso si ha un livello di trasparenza intorno al 50% - 75% nella visualizzazione dei bandeau di cambio canale, giusto per darvi un'idea)

Con defiant è possibile importare ed esportare le tabelle dei colori da alcuni tipi di file come le palette di photoshop...

I file di immagine (*.ima) sono file che contengono una singola immagine, questa può avere qualsiasi dimensione e ovviamente si deve basare sulla palette di un file lut che deve essere presente nel firmware.

Aprendo un file di questo tipo con defiant vi si aprirà una finestra dove vi verrà visualizzata l'immagine e dove sarà possibile importare o esportare l'immagine in formato bitmap. Vi verrà anche indicata la dimensione dell'immagine.

I file immagine di tipo compresso mpeg-2 (*.fnu) sarà possibile visualizzarli con defiant ma non sarà possibile fare alcun altra operazione. Infatti tali file sono semplicemente importati all'interno del firmware con la sola estensione (*.m2v) cambiata in *.fnu.
Nel caso voleste modificare questi file lo potrete fare con degli appositi programmi più sofisticati e dedicati allo scopo che permettono di decidere quanto comprimere l'immagine e quindi la sua qualità.
Un esempio è il programma TMPGEnc (ma ce ne sono parecchi altri!).

I file di tipo testo (*.mpg) sono i file forse più semplici. Vanno considerati come dei vettori su file di stringhe di testo.

Aprendo un file di testo con defiant vi verrà visualizzata una finestra dove selezionando dal menu a cascata l'indice della stringa vi verrà visualizzato il rispettivo testo e potrete modificarlo. Potrete anche aggiungere o eliminare delle stringhe intere con gli appositi pulsanti.

I file di icone (*.ico), sono delle librerie di icone che contengono appunto delle icone!!

Aprendo una libreria di icone con defiant vi verrà mostrato tutto il listato delle icone in un menu a cascata.
Sarà possibile eliminare delle singole icone, aggiungerne delle nuove o sovrascrivere quelle esistenti con delle nuove tramite l'importazione da file bitmap (esiste anche la possibilità di esportarle).
Anche qui le icone si dovranno basare su un file di lut per poter essere visualizzate correttamente.
Il nome alle singole icone verrà dato in automatico all'atto dell'inserimento della stessa all'interno della libreria tramite imamgine bitmap.

I file di pannelli (*.pan), sono delle sorte di form che vengono utilizzati nel pantalk.
In particolare sono un insieme di widget (un widget è un oggetto che può essere inserito in un pannello) che formano un output visuale, che verrà mostrato a schermo.

Vediamo come ce lo mostra Defiant. Apriamo un file di questo tipo e vi si aprirà una finestra questa finestra può presentare a prima vista molte cose un pò confusionarie e non comprensibili, ma con un pò di pratica si capirà ben presto come utilizzare al meglio ogni informazione data.

Cercherò di fare una descrizione di cosa fa e cosa non fa ogni parametro impostabile.

- Zona output: è la zona dove sono disegnati tutti i widget del pannello e varia a seconda della pagina scelta e dei widget visualizzati/inseriti. E' possibile spostarli con la semplice operazione di selezione e spostamento con il mouse tenendo premuto con il tasto destro.
- OK, Cancella: servono per salvare i cambiamenti apportati al pannello o per chiudere senza salvare nulla
- Blocca/Sblocca Widget: il pulsante con l'icona del lucchetto blocca o sblocca la possibilità di spostare gli elementi con il mouse nella zona di output, questo per evitare accidentali spostamenti degli elementi del pannello.
- Mostra schermo: In un pannello possiamo avere fino a 0x1F pagine, in ogni pagina andremo a impostare un certo tipo di output a seconda delle nostre esigenze. (per cambiare pagina selezionare la pagina voluta, non vengono visualizzate le pagine che non contengono widget)
- Dimensione griglia: impostando la slidebar possiamo decidere di quanto far spostare un widget sullo schermo in base a degli spostamenti predefiniti. Provate a impostarla su 3, vedrete che se provate a spostare un widget col mouse vi verrà spostato di minimo 3 in 3 pixel.
- Griglia di tracciamento: mettendo una spunta di check su questa casella vedremo dei puntini sullla zona di output che ci danno una traccia degli spostamenti.
- Posizioni Relative: alcuni firmware non fanno riferimento alla posizione rispetto al display a cui è associato il pannello ma a una posizione relativa al widget Panbox, questa casella va spuntata nel caso il firmware faccia parte di questa categoria (che io sappia solo i firmware sony non utilizzano in parte le posizioni relative... tutti gli altri firmware necessitano delle posizioni relative, quindi casellina con check di spunta abilitata)
- Sposta alla posione: questo pulsante vi sposterà un oggetto alla posizione indicata dal menu a cascata che si trova affianco. La posizione ne indica l'indice e quindi il momento in cui verrà disegnato su schermo, gli elementi con indice minore saranno sovrapposti da quelli con indice maggiore.
- Copia, Incolla: selezionando un widget sarà possibile copiarlo e successivamente incollarlo nella zona di output di un qualsiasi altro pannello, anche di pannelli di istanze diverse di defiant.
- Cancella Elemento: elimina l'elemento selezionato
- Clona Elemento: copia e incolla nella stessa pagina dello stesso pannello l'elemento selezionato
- Crea Elemento: crea un nuovo elemento inserendolo nel pannello. Il tipo di elemento da creare è scelto dal menu a cascata che si trova sotto al pulsante.

Nella zona di output si seleziona un elemento con il click su di esso tramite pulsante destro del mouse.

Si sposta un elemento premendo e mantenendo premuto il pulsante destro del mouse e muovendo il cursore del mouse stesso.
Alla selezione di un widget tutti i campi ad esso relativi verranno aggiornati in automatico da defiant.

Nella finestra dei pannelli troviamo a destra della zona di output una sezione divisa in pagine/tab.

Visualizziamo la pagina chiamata "Pannello" e scopriamone le caratteristiche:

- Palette: Cambiando questo parametro cambieranno i colori visualizzati nella zona di output in base alla palette scelta. Ovviamento questo è un parametro fasullo visto che la scelta della palette non fa parte del file *.pan ma serve per dare un esempio indicativo di quel che visualizzerete su schermo (diciamo un anteprima!). Le palette vanno impostate via codice oppure si mantiene quella di default dichiarata nel file *.app.
- Metodo inizializzazione: è espresso in esadecimale e indica quale è il metodo che viene preso in considerazione quando il pannello viene caricato in memoria (vedremo più avanti come interpretare meglio i metodi associati ai widget e al pannello, per ora considerateli come il codice evento che richiama il pannello in un certo momento. In questo caso all'inizializzazione)
- Metodo finalizzazione: stessa cosa del metodo di inizializzazione, solo che questo si attiva alla chiusura del pannello.
- Metodo aggiornamento: indovinate un pò? Esatto avviene quando si fa un refresh del pannello.
- Nome classe: è il nome della classe che si intende prendere come riferimento per l'associazione di variabili ai vari widget del pannello (vedremo più avanti come sarà possibile visualizzare il contenuto di una variabile tramite un widget impostandola come parametro del widget stesso). Il nome deve essere senza estensione e se si vuole utilizzare una classe di un altro modulo bisogna dichiararla in questo modo: "nome_modulo:nome_classe"
- Libreria icone: è il nome della libreria delle icone che si intende utilizzare per associare le icone a dei widget. Come per le classi il nome va dichiarato allo stesso modo: "nome_modulo:nome_libreria".
  Nel caso il modulo sia lo stesso del pannello si può evitare di inserire il nome del modulo.
- Lingua 0 .. 4: spuntando le relative caselle indicheremo quante e quali lingue supporta il pannello.
- Lingua predefinita: E' il numero della lingua che viene usata come default.
- Colore Sfondo/Seleziona Colore: Cambiando il codice del colore sarà possibile cambiare lo sfondo della zona di output. Questo è un parametro fittizio visto che come per le palette serve solo per dare una anteprima del pannello. A volte può capitare che con il nero non si vedano dei bordi fatti in nero e per questo sarà necessario cambiare colore... che so viola... vabbeh è un colore di merda... ma se vi piace potete metterlo!!
- Visualizza Sfondo MPEG-2: In questo menu a cascata verranno inseriti tutti i file all'interno del firmware che hanno estensione *.fnu. In questo modo sarà possibile visualizzarli come sfondo al posto del mono-colore. Ovviamente anche questo è un parametro fittizio e serve solo come anteprima
- Colore Griglia/Seleziona Colore: Cambiando il codice del colore sarà possibile cambiare il colore della griglia di tracciamento nella zona di output. Anche questo è un parametro fittizio visto che come per lo sfondo della zona output serve solo per dare una anteprima del pannello. A volte può capitare che con il colore bianco della griglia non si veda la griglia

stessa su alcuni widget e per questo sarà necessario cambiare colore!!

Andando nel menu configurazione di Defiant è possibile impostare varie caratteristiche per i pannelli alla apposita voce Pannelli.

- Dimensioni massime: imposta le dimensioni della zona di output. Di default è 486*576 (il minimo), ma potete aumentarle fino alla grandezza dello schermo tv (576*720, credo... boh non ricordo... comunque è limitato da defiant)
- Colore sfondo: è il colore che viene visualizzato di default alla apertura di un pannello come sfondo della zona di output, di default è nero, ma potete impostarlo come vi pare e piace!
- Elementi con posizione relativa al pannello di fondo: Ricordate il parametro posizioni relative? Beh viene considerato come se fosse checckato di default selezionado questa opzione
- Non salvare i pannelli con più di 183 elementi: serve per effettuare un controllo sul numero di elementi del pannello, infatti se si superano questa quota il pannello crea errori di memoria durante l'esecuzione del firmware. Si è lasciata la possibilità di disabilitare questo controllo perchè alcuni decoder potrebbero funzionare ugualmente anche con più di 183 elementi (io non so dirvi quali siano, ma penso che gli strong lo facciano, ecco che poi hanno anche problemi ogni tanto...)
- Colore griglia: è il colore che viene utilizzato di default per visualizzare la griglia
- Dimensioni griglia: è la dimensione della griglia presa di default quando si apre il pannello
- Griglia attiva: indica se volete che la griglia di tracciamento sia visualizzata all'apertura di un pannello per default

Selezioniamo la pagina "Elemento (visuale)".
Analizziamola:

- Elemento selezionato: è il numero dell'elemento che avete selezionato, cambiando questo numero dalla lista selezionerete il rispettivo elemento. (rappresenta la posizione dell'elemento nell'ordine di visualizzazione, vedi "sposta alla posizione")
- Tipo Elemento: è il nome del tipo di widget associato all'elemento. Ci sono diversi tipi di widget, in base al tipo sarà possibile fare certe cose e impostare certe opzioni.
- Nome Elemento: è il nome che identifica l'elemento all'interno del pannello.
- Lingua mostrata: selezionando la lingua potrete modificare i parametri dell'oggetto in base alla lingua scelta, ogni lingua può avere il parametro testo impostato in modo diverso (vedi dopo)
- Coordinate Elemento: x è la posizione nell'asse delle x, y in quello delle y, Altezza e Larghezza rappresentano... beh che vi devo spiegare anche questo?!
- Font Elemento: Potrete selezionare il tipo di font, la dimensione e se allineare il testo a destra, a sinistra o mantenerlo centrato.
- Stile Elemento: Rappresenta lo stile che verrà mostrato. E' possibile associare ad ogni widget più stili che verranno impostati come avviene spesso durante la selezione di un widget con il cursore. In ogni stile è possibile definire il tipo di bordo (Linea semplice, Linea arrotondata, smussatura incassata, smussatura in rilievo, oppure la composizione/unione di questi ultimi due tipi di bordo... provateli tutti per farvi un'idea). Potrete impostare le larghezze dei bordi, nel caso dei compositi imposterete due larghezze essendo la composizione di due tipi di bordo.

La mini-sezione "colori" rappresenta i colori del testo, del bordo e di sfondo (i colori saranno visualizzati in base alla scelta della palette nel campo apposito nella pagina/tab "Pannello").
- Visibilità su schermi: cliccando su questo pulsante vi si aprirà una finestrella dove potrete spuntare le pagine del pannello in cui visualizzare l'elemento selezionato.
- Testo (non è sempre presente): è una casella dove potrete scrivere il testo che l'elemento dovrà visualizzare. A seconda del tipo di elemento verrà interpretato il testo come un listbox o come semplice stringa, ecc...
  Questa casella è presente in tutti i widget a parte nel SubPanel, nella IconBox e nella Window.
- Il pulsante con l'immagine del dado (non è sempre presente): vi serve per vedere i caratteri del font scelto. In questo modo potrete inserire nella casella testo anche i caratteri speciali. Questo pulsante è strettamente legato alla casella Testo.
- Pannello (non è sempre presente): viene visualizzato solo se si seleziona un elemento di tipo SubPanel. Dovrete inserire il nome del pannello che desiderate passare come parametro.
- Icona (non è sempre presente): viene visualizzato solo se si seleziona un elemento di tipo IconBox. Qui dovrete inserire il nome dell'icona scelta dalla libreria icone impostata preventivamente nella tab/pagina pannello.
- Immaigne (non è sempre presente): viene visualizzato solo se si seleziona un elemento di tipo Window. Qui dovrete inserire il nome dell'immagine da visualizzare all'interno del widget. Il nome immagine è il nome di un file immagine (*.ima) ma con estensione diversa (*.rf4). Non ne ho mai capito il motivo, quindi non me lo chiedete!

A questo punto apriamo la pagina che si chiama "Elemento (altro)".
Analizziamo anche qui i vari campi:

- Chiave: Selezionando dal menu a tendina una chiave associeremo il verificarsi di un evento alla pressione di un tasto del telecomando relativamente al widget.
- Metodo Elemento: è il codice dell'evento che viene associato al widget.
- Priorità: la priorità al verificarsi dell'evento rispetto ad altri.
- Gruppo Widget: questo campo è utilizzato per le funzioni di gruppo (esempio Set_Panel_Value($CURRENT, $WIDGETS_ACTIVE_ONLY_GROUP_GE, 0x2);)
- Variabile: è il nome della variabile associata al widget. (potremo in questo modo visualizzarne il contenuto, come avviene per esempio con le variabili di tipo testo)
- Indice dell'array: è l'indice del vettore (nel caso esista) riferito alla variabile
- Bit di stato: abilitato (utilizzato per rendere abilitati i button), Sconosciuto (lo dice il nome... non si sa che fa), Multilingua (nel caso sia un widget multilingua), Esci dal pannello (nel caso si debba chiudere il pannello una volta associato l'evento, ma questo bit non mi è molto chiaro come funzioni... non lo ho mai utilizzato), Bitmap (nel caso contenga immagini), Nascosto (se non si vuole visualizzare il widget), Non disegnare (il widget non verrà disegnato su schermo, da precisare che è diverso dallo nascondere!), Sempre zero (boh!).
- Pannello succ.: Pannello successivo (io non lo ho mai utilizzato), sembra esegua un pannello una volta che si è scaturato l'evento del widget.
- Posizione: è la posizione di riferimento del widget rispetto al pannello

Avendo parlato dei pannelli siamo costretti a fare una piccola escursione sui **tipi di widget** che un pannello può possedere:

- Panbox: questo oggetto è l'oggetto principale del pannello, serve per contenere tutti i widget

al suo interno. Essendo l'oggetto principale ce ne è solo uno (state attenti che defiant non applica controlli sul numero di panbox, potreste avere qualche problemino di visualizzazione se provate a caricare un firmware con più panbox). Questo elemento deve sempre essere l'elemento che si trova alla posizione 0.

- Button: beh, il nome dice più o meno tutto, è un bottone e come tale gli si può associare un evento che può avvenire alla pressione di un tasto chiave (quelli del telecomando tanto per intenderci)
- InputBox: Attraverso una serie di impostazioni permette di prendere in input del testo, inserito tramite telecomando. Il tipo di dati che verrà inserito può essere stabilito attraverso codice e si possono anche inserire delle mask attraverso dei semplici trucchetti. Può essere associato a una variabile di tipo alphanum (255 caratteri).
- TextBox: Beh... anche qui... è un box di testo, se volete visualizzare del testo usate questo, spesso in MHW viene utilizzato per creare anche delle forme rettangolari con le varie impostazioni arrotondate per la creazione dei bandeau.
- Frame: una sorta di pannello divisore tridimensionale, in MHW viene utilizzato poco serve per evidenziare qualche area.
- SPanel: boh... mai utilizzato...
- ListBox: serve per mostrare dei listati di testo, è possibile nei listbox far scorrere il cursore (per cursore intendo la selezione degli elementi) sugli elementi della lista stessa
- EnumBox: in questo elemento si possono inserire delle liste di testo che vengono considerate come dei dati di tipo enum, la selezione dell'elemento successivo avviene automaticamente durante la pressione dei tasti freccia associandoli correttamente.
- Counter: Visualizza il valore di una variabile intera (io non lo ho mai utilizzato e mai visto utilizzare in MHW...)
- SubPanel: è come un pannello solo che deve essere di questo tipo quando è inserito in un altro pannello, si comporta alla stessa maniera del pannello.
- CheckBox: Permette di fare un segno di spunta sulla scelta nel caso venga selezionato.
- RadioBox: un widget che viene utilizzato per dare la possibilità di marcare solo una opzione tra le tante proposte
- BarGraph: viene utilizzato per visualizzare l'andamento di un processo che necessita di un riscontro visuale del suo punto attuale di lavoro... viene utilizzato spesso per trasferimenti di file, caricamenti ecc...
- IconBox: viene utilizzato per visualizzare le icone delle librerie di icone.
- EditBox: Permette di editare il testo sfruttando alcune specifiche funzioni dedicate agli editbox.
- Window: viene utilizzato per visualizzare i file di tipo ima.
- Slider: beh dovrebbe (dico dovrebbe perchè non lo ho mai utilizzato) essere simile agli slider che si usano nei programmi visuali.
- Panel: Un oggetto di tipo pannello, utilizzato in MHW come il frame.
- PoList: ??? Forse una sorta di lista

Una piccolissima precisazione... questi dati sono stati ricavati da quanto propone Defiant. Non esiste documentazione alcuna riguardo tutti questi elementi. La maggior parte di essi sono stati ricavati attraverso una sorta di utilizzo di bruteforce dei pannelli. Probabilmente molti di questi oggetti non li utilizzerete mai, e probabilmente alcuni non sono supportati appieno ne in MHW, ne da Defiant stesso.

Se qualcuno avesse qualche informazione in più può tranquillamente esplicarla che non si offende nessuno

Ultima cosa... il brute force sui pannelli è stato fatto con un programma che si chiama HPanel. HPanel è il programma originale della ditta HyperPanel che permette di modificare e lavorare sui file scritti in Pantalk.
L'unico nostro problema è che in MHW non si lavora in Pantalk puro e spesso alcune cose possono risultare diverse. Senza contare che questo programma non si trova da nessuna parte se non comprandolo, e poi serve tutta una serie di licenze dove tempo fa bisognava dimostrare di essere sviluppatori su sistemi embedded che sfruttavano il pantalk per poter comprare questo programma, ecc...
Utilizzando un programma creato da JimmiF si riuscì a convertire i file pantalk originali in pantalk MHW. Questo però sempre tramite vie più o meno reali, capitava infatti (spesso) che alcuni elementi non venissero visualizzati correttamente su decoder. Probabilmente non sono nemmeno previsti nel MHW. Purtroppo queste sono le poche informazioni che ho in merito.

Ad ogni modo come già detto non vi dovete preoccupare perchè per fare un buon firmware in MHW non c'è bisogno di neanche la metà dei widget che esistono!

Siamo arrivati all'ultimo punto cruciale dell'esecuzione del MHW... gli script (file con estensione *.cpi)!

Finalmente è possibile avere un compilatore degli stessi anche in defiant, visto che mai nessuno si è prodigato in tale opera e mai nessuno ha offerto il suo codice per il bene della comunità intera affinchè fosse integrato in defiant (vorrei ricordare che sebbene defiant sia un tool iniziato per volontà di sa1vad0r e attualmente è stato migliorato da me, molto codice scritto da sa1vad0r è stato scritto perchè c'era gente che offriva codice e conoscenze gratuite a tutti, purtroppo tale ideologia è andata persa con il tempo e per questo motivo defiant è rimasto per lungo tempo fermo alla versione 2.62 con le varianti 2.64/2.65 ognuna diretta da diverse persone ognuna con diverse caratteristiche per diversi firmware... non si capiva più una mazza, non esisteva più la logica di uscita delle versioni di un programma e a defiant mancavano ancora molte informazioni sulla corretta modifica dei file di classi, dei pannelli e tante piccole cose... io ho solo cercato di riunire tutte le mie conoscenze in defiant rifancedolo essere quello che voleva diventare... IL tool di sviluppo per il MHW... finalmente sono anche riuscito a fare il compilatore, migliore di tutti... ihihi).

Non esistendo un compialtore interno inizialmente in defiant era presente solo un decompilatore molto scarno e parecchio "buggoso"...
Col passare delle versioni ho cercato di migliorarlo aggiungendo le mnemonics e migliorando la decompilazione in generale fino a farlo diventare migliore di uComp.

Ovviamente non essendoci inizialmente un compilatore ho optato per una mezza integrazione con HPanel e uComp.
Per l'integrazione del compilatore con ucomp e HPanel vi riporto al thread Piccoli GRANDI tool di sviluppo.

Impostando nel modo corretto defiant è possibile compiere l'atto della compilazione in pochi semplici passaggi automatizzati.

Vediamo come: selezionare dal menu Configurazione->Compilatore *.cpi->Usa compilatore interno quando andrete ad aprie un file di script vi verrà aperto un editor che utilizzerà il compilatore integrato in Defiant.
Selezionando Configurazione->Compilatore *.cpi->Compilatore Interno->Mnemonics Attivati vi verrà mostrato il codice decompilato con l'uso dei mnemonics al posto dei valori numerici (molto utile per capire il significato di alcuni parametri)
Selezionando Configurazione->Compilatore *.cpi->Modifica colorazione... potrete modificare la colorazione delle parole chiave del linguaggio di compilazione.
Nel menu Configurazione->Script->Font... andrete a modificare il Font di default dell'editor di script.
Nel menu Configurazione->Script->Gutter nascosto deciderete se visualizzare di default il gutter nell'editor di script.
Nel menu Configurazione->Script->visualizza finestra editor massimizzata vi visualizzerà all'apertura di uno script la finestra dell'editor di script massimizzata.

Per compilare lo script cliccate su “compila modifiche” e cliccando su OK salverete le modifiche effettuate se non ci saranno errori. Se sono presenti errori (visualizzati nella finestra di log in fondo) non verrà salvata alcuna modifica. Se cliccate su cancella verrà chiuso lo script e nessuna modifica verrà salvata.
Cliccando su “compila ed elimina writetraces” verrà compilato lo script eliminando i writetraces.
Cliccando su “compila ed elimina commenti” verrà compilato lo script eliminando i commenti.
Cliccando su “memorizza modifiche” verranno salvate le modifiche applicate allo script una volta compilato correttamente (senza errori) e verrà visualizzato lo script aggiornato alle modifiche senza dover chiudere e riaprire lo script (questa azione effettua il salvataggio dello script come l'OK ma non chiude lo script)

Vorrei ora dedicare due o tre righe in generale per quel che riguarda gli script.
Gli script sono un insieme di codice che è scritto in pcode, essendo pcode ognuno potrebbe personalizzarlo come vuole! Per questo motivo in defiant è presente un file che contiene la dichiarazione di tutto il codice in pcode affinchè il decompilatore interno possa interpretarlo nel modo corretto, volendo si possono cambiare i nomi alle singole funzioni, ma si è sempre cercato di mantenere una struttura simil-c affinchè fosse comprensibile a quasi tutti senza dover memorizzare molto di nuovo rispetto ad altri linguaggi. Il vero Pantalk è quello che viene utilizzato da HPanel ed è un misto tra C, VB e Pascal. Insomma un mezzo macello, io preferisco molto di più la struttura simil-C che rende le cose più semplici anche da un punto di vista di come è strutturato il linguaggio.

Gli script sono divisi in due categorie.
Indipendent script e Panel script.
I primi sono gli script singoli che non dipendono da nulla, i secondi, sono gli script che sono associati ai pannelli e devono avere lo stesso nome del pannello (questi ultimi non possono essere richiamati dagli indipendent script, invece è possibile fare il contrario).
Ricordate quando si parlava di metodi ed eventi del pannello o dei widget?
Ecco... cercate di ricordarlo o ritornate a darci uno sguardo... nei panelscript avviene l'associazione all'evento tramite l'istruzione Start_Method(0xYYYY);
e termina con End_Method(0xYYYY);
Dove 0xYYYY è un numero esadecimale che indica il numero del metodo.
Se ad esempio associate ad un Button l'evento 0x005E (numero preso a caso) quando andrete a premere il pulsante chiave che avete dichiarato nel campo Chiave, verrà eseguito il codice che è

presente tra start_method ed end_method (nel panelscript con nome uguale al pannello) rispettivamente al numero di evento.
uComp e defiant aiutano in questo indicando quale elemento sfrutta l'evento dichiarato.

In questi due paragrafi spiego un pò come utilizzare i vari compilatori:
Piccoli GRANDI tool di sviluppo.
HPanel vs Defiant

Piccoli GRANDI tool di sviluppo.

Dunque, inizio questo paragrafo dove mi piacerebbe riunire una manciata di tool storici che sono serviti in passato e che a volte possono ancora servire.

Iniziamo con uComp.
Che roba è?
E' un compilatore/decompilatore per i file di script MHW.
E' stato creato da pGete (un tizio spagnolo che sviluppava sugli strong, ora pare si sia ritirato) e ha la particolare caratteristica di essere molto funzionale.
Utilizza una sintassi simil-C per trasformare il codice scritto in P-Code per essere poi interpretato dal firmware.
C'è anche l'opzione per vedere la sintassi Pantalk originale, ma non permette di compilare in questo modo.
La decompilazione avviene per tutti i file. Basta passarli come parametri il programma provvede da solo a decomprimerli e a decompilarli in formato testo.
Creerà vari file.
*.var per la decompilazione delle variabili.
*.ccc decompilazione degli script.
*.cod codice in PCode
*.pas decompilazione dei pannelli
ecc...

Aprendoli con un normale editor di testo è possibile vederne il contenuto e modificarlo, successivamente ripassando i file per ucomp sarà possibile ricompilare il tutto.
La faccenda è particolarmente macchinosa perchè essendo un programma console-based non è proprio l'ideale per un utente che vuole fare le cose senza dover ogni volta stare a pensare che comando scrivere.
Per questo motivo pGete in primis ha creato dei plug-in da usare con editplus.
Inoltre in Defiant ho inserito la possibilità di interfacciare ucomp con defiant stesso. In questo modo con un semplice doppio click sul file sarà possibile decompilarlo o compilare il file una volta modificato.

Ovviamente queste opzioni sono tutte impostabili in defiant e nei plugin di editplus.
Riguardo le istruzioni di ucomp notiamo che richiamandolo possiamo visualizzare tutto il listato dei suoi parametri.

Tempo fa li avevo tradotti in inglese e adesso li riporto (nel programma sono in spagnolo!)...

-s read directoryes and generate file.ccc,file.aps,file.var,file.pas,file.dat',0

-S integrate .cod into .ccc',0
-c read file.cod and create modified .cpi (-C all)',0
-b read file.ccc and create .cod ',0
-a read file.aps and create modified .app (-A all)',0
-v read file.var and create modified .cla (-V all)',0
-d read file.dat and create modified .dat (-D all)',0
-p read file.pas and create modified .pan (-P all)',0
-s0 delete Class. not needed into Pans',0
-u Save files decompressed',0
-U read file decompressed',0
-x replace actual files (for default into NUEVO)',0
-k create file.crc with a list of files and crcs into directoryes'
-g create file.cfg with a list of files into directoryes',0
-h show bytes when generate sources',0
-I Pantalk sintax (default c)',0
-i show pilas when generate sources',0
-o show comments into sources',0
-O show C into code',0 ;
-n Strong',0
-f show hex functions',0
-t show types together variables' name',0
-j create file.f u n with all calls to functions',0
-r create references into log',0
-F create file with calls to functions',0
-# mantains numerics constants',0
-B create file.cod by the file.ccc and file.var',0
-q only messages of warning and error',0
-Q comments improvements into the code',0
-W delete comments into cpis',0
-w delete write_traces into cpis',0
-WXXXX omit messages warning XXXX',0
-EXXXX omit messages error XXXX',0

Ovviamente è possibile fare l'associazione di più parametri per ottenere il risultato voluto.

Ricordate solo una cosa. Uno script di per se è un file unico, ma al suo interno contiene i riferimenti alle variabili tramite la classe. Per questo motivo è necessario decompilare prima le classi e successivamente gli script per ottenere una decompilazione corretta.

Detto questo non affidatevi sempre troppo a ucomp, spesso sbaglia in alcune decompilazioni per questo motivo io preferisco usare defiant che è diventato efficientissimo e avendolo fatto io, so quello che combina :D
Nel prossimo paragrafo farò una spiegazione breve sull'utilizzo del plugin per editplus e di defiant con ucomp.

I sorgenti pGete non li ha mai resi pubblici e quando glieli ho chiesti ha detto che non li aveva più essendosi ritirato ha cancellato ogni cosa, peccato si poteva cercare di migliorarlo essendo già una buona base di partenza. Vabbeh poco male...

Ecco... la ho trovata, una piccola guida sul primo utilizzo di ucomp con editplus che avevo fatto insieme a porksfree e pubblicata in un vecchio documento.

Modificare firmware con uComp di pGete (a cui vanno i nostri ringraziamenti).
Questa non vuole essere un manuale esaustivo delle potenzialità di uComp, ma solo una guida introduttiva al suo utilizzo.
Cosa occorre:
- Defiant
- uComp.exe
- plug-In di uComp per Edit Plus
- EditPlus

Defiant servirà per esportare ed importare i file da modificare e compilare (in modalità non compressa).
EditPlus e Plug In x uComp, costituiscono lo strumento per modificare i sorgenti (script, .cpi, e variabili, .cla).
uComp non verrà utilizzato 'direttamente' (per voi sarà uno strumento 'trasparente'), mettetelo in una directory coperta dal path di sistema (windows o system32) e dimenticatelo (per il momento).

Vantaggi di uComp rispetto all'accoppiata panparser/Hpanel (o il successore PCBootDirect):
- praticità di utilizzo
- velocità di modifica/compilazione
- semplicità di configurazione ed apprendimento
- grazie all'utilizzo delle costanti mnemoniche allo highlighting delle parole chiave rende leggibile e comprensibile il codice
- si può utilizzare un vero e proprio editor invece di quell'accrocco fornito con Hpanel (oddio, si può fare anche con Hpanel, ma con salti mortali)
- non scasina .cpi collegati ai panel (ogni .pan ha un .cpi che gestisce gli eventi legati ai widget ivi contenuti), cosa che invece fanno panparser/Hpanel e vi obbligano a rimettere le cose a posto con un editor esadecimale.

Svantaggi:
- non supporta tutte le api del 9701 ed in alcuni casi dà problemi, in questo caso utilizzare Hpanel
- alla fine non è una vera e propria suite di sviluppo e quando i file da tenere sott'occhio incominciano ad essere molti, si sente la necessità di qualcosa di più completo

Api non supportate e problematiche:
- proc_0305 ()
- proc_02E8 ()
- proc_02E5 ()
- proc_0258 (Get_Pmt_Info)

Queste api si trovano in pochi script, ecco alcuni:
- basic\init1.cpi
- basic\pmt_sct2.cpi
- basic\zap_con1.cpi
- basic\zap_con2.cpi
- s_scan\scanning.cpi

- s_scan\mtpx_sca.cpi
- altri ? Boh! Se li trovate, fatelo sapere

In questi casi usate Hpanel/panparser, ammesso che riusciate ad avere dei file di configurazione completi di tutte le api, per Hpanel/panparser oppure c'è un trucchetto:
con un editor esadecimale,aprite il .cpi (esportato in modo non compresso) cercate il codice della proc problematica. Esempio: proc_02E8 -> 02E8.
Sostituitelo con una conosciuta e che utilizza lo stesso numero e tipo di parametri. Fate il vostro lavoretto con editPlus/Ucomp, compilate e poi rifate la sostituzione al contrario (sempre su file non compressi).

Veniamo al dunque. Come fare…
Innanzitutto, potete lavorare esportando tutto il fw o solo quello che effettivamente vi serve.
Il primo caso è comodo per avere una visione di insieme, per fare delle ricerche su tutto il codice, magari per valutare gli impatti di una modifica.
Noi lo sconsigliamo per il lavoro corrente, perché ci sarebbero tanti messaggi terroristici (warning ed errori) in compilazione/decompilazione che possono fuorviare.
Il secondo è altamente consigliabile, specie nei primi tempi, per le modifiche vere e proprie.
Procediamo.
- Installate il sw: Ucomp ed EditPlus, il primo lo potete mettere dove già accennato oppure nella cartella di EditPlus, insieme ai plug-in.
- Create una cartella di lavoro (es: C:\MIO_FW)
- Aprite il fw con defiant, posizionatevi nella cartella radice (quella che porta il nome del fw)
- premete l'icona "estrai il nodo selezionato senza decompressione"
- vi ritroverete una cartella con il nome del fw in C:\MIO_FW, al suo interno tutti i moduli del fw.
- cambiate il nome della cartella con il nome del fw, in qualcosa di + semplice (ma non è necessario), esempio mioFW.
- create un file di testo vuoto nella cartella c:\MIO_FW e chiamatelo come la cartella creata da defiant e da voi modificata, ma con il suffisso .ccc, es: mioFW.ccc.
- aprite il file vuoto con EditPlus e premete CTRL+1 (oppure dal menu: tools->Genera C files, per attivare la macro di decompilazione), se tutto è stato fatto per bene avrete il sorgente del vostro FW, insieme a 1000 messaggi di errore e di warning!!! Non vi preoccupate. UComp, oltre al file mio_fw.ccc, crea altri file, tra cui mio_fw.var. Qui sono contenute le dichiarazioni delle variabili ovvero i .cla. Se volete modificare/aggiungere variabili fatelo qui.
Per compilare e generare i file modificati premere CTRL+2 (oppure tools->Compila modificationes), verificare che nella finestra di output di EditPlus non ci siano errori, oppure che quelli restituiti non costituiscano un problema (qui ci vuole un po' di esperienza) ed in seguito proseguite con la generazione dei file tramite la pressione di CTRL+3 (tools->graba modificationes). A questo punto non vi rimane che importare i file modificati nel fw con defiant (in modalità senza compressione).
Altre modalità di compilazione sono:
CTRL + 4 : compila eliminando i WriteTrace (sono le procedure che permettono di scrivere messaggi di debug nel FW, causano rallentamenti nell'esecuzione del codice, si è soliti eliminare le chiamate superflue nelle versioni definitive)
CTRL + 5 : compila eliminando i commenti
CTRL + 6 : rigenera tutti gli script

Ricordiamo che CTRL+2 e CTRL+3 lavorano solo sulle modifiche effettivamente fatte e non su

tutti i file.

Ripetiamo il consiglio di lavorare solo sulle porzioni di codice che vi interessano, esportando con defiant i moduli .app, .cpi e .cla collegati alle modifiche che volete fare. E' più semplice. Poi, man mano che modifichiamo il fw aggiungiamo roba.

Facciamo un esempio. Vogliamo modificare i file basic\init2.cpi e basic\bad_ecm.cpi
1) creiamo la cartella c:\mio_fw

2) individuiamo le variabili (.cla) istanziate nei file da modificare aprendo lo script con defiant. Nel nostro esempio:
in bad_ecm.cpi si usano:
class basic;
class global;
class table;
class card_evt;
in init2.cpi invece:
class global;
class basic;
Bene, già sono utilizzate nel primo script. Vediamo in quali moduli (cartelle) sono situati i .cla (si fa con gli occhi). Allora:
- basic.cla, table.cla si trovano nella cartella 'basic'
- global.cla in 'commun'
- cardevt.cla in 'cardevt'
quindi tre cartelle diverse.

3) creiamo queste 3 cartelle in c:\mio_fw

4) già che ci troviamo, creiamo pure il file di testo vuoto: c:\mio_fw\mio_fw.ccc

5) in defiant selezioniamo ed esportiamo senza decompressione in c:\mio_fw\basic\ i file:
- basic\init2.cpi
- basic\bad_ecm.cpi
- basic\basic.cla
- basic\table.cla
- basic\basic.app *** importante *** anche se non lo modifichiamo, serve a uComp

facciamo lo stesso (c:\mio_fw\commun\) con
- commun\commun.app
- commun\global.cla

e (c:\mio_fw\cardevt\)
- cardevt\cardevt.app
- cardevt\cardevt.cla

6) apriamo mio_fw.ccc con EditPlus, generiamo il codice (CTRL+1), modifichiamo (tappete, tappete, tappete), ricompiliamo (CTRL+2), rigeneriamo i file (CTRL+3)

7) riprendiamo dalle rispettive cartelle i file modificati ed importiamoli nel fw con defiant (senza compressione).

Per utilizzare Defiant con uComp dovrete fare qualche piccolo e semplice passaggio.
Mettete uComp.exe dove più vi aggrada (wow che parolone, sembro un avvocato...)

Aprite defiant e andate nel menu Tools-> Settings.
Vi si aprirà una finestra dove andrete a impostare la directory in cui si trova ucomp e poi ci sono dei parametri chiamati:
- decompilazione
- compilazione
- No commenti
- No write traces
- memorizzazione modificazioni.

Questi parametri sono impostati di default uguali (o simili) a quelli impostati per i plugin di editplus. In questo modo potrete sfruttare le opzioni di decompilazione, compilazione come vi pare e piace.
Cliccando sui tre pulsanti di fianco all'impostazione vi si aprirà una finestra dove potrete avere una visuale completa delle impostazioni di ucomp.

Piccolo OT:
defiant lo ho realizzato affinchè fosse dinamico nel caso si presentassero altre versioni di ucomp o compilatori simili.
E' possibile tramite i file ini parametrizzare ogni singola opzione riguardante il compilatore come per l'appunto le opzioni disponibili in ucomp.
Magari se dovesse interessare a qualcuno in questo modo dovrebbe sapere come/cosa fare.
Fine OT.

Finito cliccate su Ok e vi memorizza tutto quanto.
Andate ora nel menu Configurazione->Compialtore *.cpi e selezionate 'Usa uComp'.
Adesso avete impostato uComp come decompilatore/compilatore che verrà utilizzato con defiant all'apertura dei file di script.

Non è finita qui... ehehhe... che vi credete... se selezionate dal menu Configurazione->Compialtore *.cpi la voce 'Modifica Colorazione...' andrete ad impostare tutti i parametri e i colori che verrano evidenziati a dovere dall'editor degli script presente in defiant. In questo modo ognuno può personalizzare la propria colorazione (ovviamente quella di default è già presente nel pacchetto defiant e la ho fatta in base ai miei gusti)

Avendo lasciato le impostazioni di default per i parametri di ucomp, all'apertura del file di script avremo il nostro bello script decompilato e una volta che avete fatto le vostre modifiche dovrete cliccare sui pulsanti vicini al tasto cancella che eseguono i compiti impostati nelle parametrizzazioni nel menu settings.
Quindi fate le vostre modifiche e premte Compila (il primo tasto), poi se non avete errori vi si abiliterà Memorizza e cliccandoci sopra vi verrà salvato nel firmware il file compilato correttamente.
Se volete risparmiare spazio nel firmware compilate con l'opzione elimina writetraces o elimina

commenti in base a ciò che volete fare. (da premettere che con la versione 3.0 elimina commenti funziona male perchè ucomp per qualche strano motivo scasina tutto lo script... bah... misteri di ucomp)

Adesso vorrei parlare del PanParser (creato da JimmyF, ma si firma Smotritel, in Java) che è stato forse il primo utile strumento per la compilazione e la decompilazione degli script ma anche di tutti i file MHW. Infatti il PanParser ha il compito di convertire i file MHW in patalk puro e viceversa.

Vediamo come utilizzarlo:
Per utilizzare tale programma dovete avere installato le librerie Java Runtime 1.2 o superiori. PanParser, sono una raccolta di librerie Java che servono per “decompilare” i file .pan, .cla, .isc, che abbiamo decompresso tramite defiant o manualmente tramite i programmi pack/unpack (o in qualsiasi altro modo conosciate). Per funzionare correttamente dovete creare un file.bat che vi permetta di richiamare tali librerie in modo semplice e veloce.
Vi riportiamo quello normalmente usato;
Notate bene che se installate il PanParser e il JDK in altre directory dovrete correggere il parser.bat.

```
echo off

rem Set the pathes before use!

SET JAVA_HOME="C:\jdk1.3.1_02\jre"
SET APP_HOME="C:\PanParser0.13"


FOR %%B IN (*.lu?) DO SET PALETTE=%%B
echo on
%JAVA_HOME%\bin\java -classpath %APP_HOME%/classes panparser.PanParser %1
%PALETTE%
pause
```

Come si può vedere il batch è molto semplice, si legge prima il file.lut (che è la palette del menu, potete anche non estrarla dalla directory da dove avete preso il .pan) e poi richiama il compilatore java, esteso con la classe panparser.
Se volete associare i file al bat, dopo aver editato il parser.bat cliccate sul .pan che volete modificare; il sistema operativo vi chiederà con quale programma lo volete aprire, voi selezionare il .bat. Da qui in poi, non servirà più fare questo.
Il risultato dell’uso del parser è un file esempio_.pan e un nomedirectory.mod.

Apportate le modifiche del vostro caso per quanto riguarda le directory dei parametri SET JAVA_HOME e SET APP_HOME.

In allegato metto il panparser ultima versione e la sua ultima patch.

JimmyF ha creato un altro semplice programma, sempre in Java, che permette di visionare i "file grafici" (esempio i pan, i lut, gli ima, ecc).
Mi sembra doveroso mostrarvi anche questo, ma come avrete capito non è comodo come utilizzare Defiant (che fa le stesse identiche cose!), ma almeno capirete il perchè della necessità di defiant e di come un tempo era macchinoso fare tutte ste cose a mano (anche la semplice estrazione dei file

avveniva a mano inizialmente, poi furono creati migliaia di programmi e defiant li ha riuniti tutti quanti).
Anche questo programma, come il precedente necessita delle librerie Java Runtime 1.2 o superiori.

Vediamo come configurarlo:
anche qui dovete utilizzare un file .bat (ImageViewer.bat) che sarà ad esempio così impostato:

```
echo off

rem Set the pathes before use!

SET JAVA_HOME= “C:\jdk1.3.1_02\jre”
SET APP_HOME= “C:\ImageViewer”

FOR %%B IN (*.lu?) DO SET PALETTE=%%B
echo on
%JAVA_HOME%\bin\java -classpath %APP_HOME% imageviewer.ImageViewer %1
%PALETTE%
pause
```

Apportate le modifiche del vostro caso per quanto riguarda le directory dei parametri SET JAVA_HOME e SET APP_HOME.
Se volete fate doppio click su un file di tipo grafico, il sistema vi chiederà con quale programma lo volete aprire, voi selezionare il .bat. Da qui in poi, non servirà più farlo.

Pan/Upan
Con questo programma per dos potete decompilare/compilare i file .pan. Come per Pack/Unpack questo programma è diviso in due e per utilizzarlo bisogna passargli i file di input e output che volete utilizzare.
Esempio:
vogliamo decompilare un file esempio.pan: Upan esempio.pan esempio_.pau
vogliamo compilare lo stesso file (esempio_.pau): Pan esempio_.pau esempio.pan

Ultimo tool di tutta questa categoria (vorrei ricordare che di tool ce ne sono parecchi, anzi quasi infiniti, infatti ogni sviluppatore si creava un proprio programma dedicato al suo decoder affinchè si facilitasse il lavoro poi sono giunti questi strumenti che io considero i più utili e quelli migliori perchè oltre ad aver fatto storia sono sempre stati la base per i principali sviluppi di firmware, col tempo poi è nato defiant e questi tool sono diventati quasi inutili), è il MHW ToolKit.
Principalmente credo volesse essere quello che è Defiant, un programma tutto fare ma è partito con la base di decompilare gli script e avere un minimo di compilatore. Questo è stato il primo vero compilatore simil-C presentato pubblicamente.
Da qui poi è nato uComp...
Il creatore (Deemonru) ha però compiuto il lavoro a metà. Secondo me un pò per le difficoltà pratiche nella realizzazione dello stesso, un pò perchè col passare del tempo è nato uComp che si è dimostrato superiore.

Aprendo questo programma possiamo notare che è molto semplice, la cosa principale che fa è

decompilare uno script in linguaggio pantalk/simil-c oltre a dare in output anche un file decompilato in PCode. La compilazione sebbene sia presente può essere fatta solo se si scrive il codice in PCode. Purtroppo la parte di compilazione simil-C non è mai stata implementata. Guardando i file decompilati in simil-C noterete che non vi è presente alcun mnemonics, la decompilazione è completamente scarna di riferimenti. Essendo stato però il primo compilatore è stato un gran passo in avanti. Si poteva fare affidamento finalmente su qualcosa che non fosse HPanel.

Con MHW toolkit è anche possibile decompilare le classi, ma è un metodo un pò spartano e cerca di ricordare un pò defiant nella gestione delle stesse, solo che per me non è stato fatto bene e si capisce anche poco come utilizzarlo senza aver fatto prima un bel pò di pratica.

Giusto per la cronaca se qualcuno lo vuole provare la decompilazione dei file avviene nella cartella principale del programma stesso, bisogna mettere gli script da compilare nella cartella che verrà appositamente creata dal programma e la decompilazione sarà riportata in un'altra cartella sempre creata dal programma.

HPanel vs Defiant

HyperPanel
HPanel è il programma ufficiale che viene distribuito dalla casa software che ha inventato il Pantalk. Permette di sviluppare diverse applicazioni e di modificare ogni cosa che è modificabile, compresa l'aggiunta delle API.

"CiccioCB" Wrote:

Installare Hpanel in D:\sat\HpDevSuite; osservare la struttura : ci dovrebbero essere 3 directories direct, exe e user.
9) Creare una directory chiamata log sotto D:\sat\HpDevSuite; in questa dir Hpanel scriverà dei files di
log utili in caso di errore.
10) Aprire adesso il zip del service pack di PanParser e copiare il file emb4p.prt nella directory D:\sat\HpDevSuite\user\data ed il file scriptranscompiler.class in D:\sat\panparser\classes\panparser.
11) Nella directory D:\sat\HpDevSuite\user\data fare una copia del file stddemo.prj chiamandolo crt.prj
12) Editare crt.prj e modificare la sezione general nel modo seguente :
IMPORTANTE : usare Ultraedit in quanto il file ha una struttura unix che il notepad non sa gestire.

[general] # Global project parameters.
vm=emb2 # VM profile (USER::data:*.vm file).
debug=no # Enable PanTalk pcode debugging.
module= # VM Toolkit default module file.
panel= # VM Toolkit default panel file.
class= # VM Toolkit default class file.
script= # VM Toolkit default script file.
inoltre cancellare delle linee dalla sezione mod e lasciare solo :
[mod] # Project modules list.

module=DEMO::javacard:javacard.mod
module=USER::data # User data (projects,volumes,etc...).
module=EXE::exe # Executable and run.ini location.
module=LOCAL::log # Trace/message file location.
13) Fare una copia del file emb.vm chiamandolo emb2.vm
14) Editare emb2.vm e modificare la sezione general nel modo seguente :
[general] # General information.
version = 5.1.15 # Version.
executable= emb # Executable.
libproc=emb4p # PanTalk procedure table.
mnemonic = embm # PanTalk mnemonic table.
log = 1,"",0,0 # Console,logfile,logsize,type.

15) Cancellare tutte le directory presenti in D:\sat\HpDevSuite\user\demo lasciando soltanto la directory
javacard; cancellare inoltre tutti i files presenti in javacard (al limite farsi una copia prima di cancellare). Riassumendo : lasciare solo la directory user\javacard e vuota.

Lanciare adesso Hpanel (run.exe nella directory D:\sat\HpDevSuite\exe)
IMPORTANTE : il programma funziona bene solo a 1024*768
1) In Module name inserire toolkit oppure cliccare su OK.
2) Ciccare su Project e aprire il progetto crt.prj.
3) Cliccare su Class.
4) Click su File, su Open, su crt.prj , su DEMO::javacard e su basic_.cla.
5) Click su Confirm ; tutte le variabili sono visibili.
6) Click su Save - questo aggiunge delle informazioni necessarie al file.
7) Chiudere la finestra Class.
8) Click su PanTalk.
9) Click su File, su Open, su crt.prj, su DEMO::javacard e, su iso_639_4.isc.
10) Click su Confirm ; il file é visibile.
11) Click su Save - questo aggiunge delle informazioni necessarie al file.
12) Chiudere la finestra PanTalk.
13) Riaprire la finestra PanTalk e riaprire il file iso_639_4.isc.
14) Sulla parte sinistra della finestra cliccare su iso_639 sotto method(1).
15) Il testo del file sarà visibile.
16) Cliccare su Debug"; se tutto è OK non ci saranno messaggi d'errore.
17) Cliccare quindi su Compiler; vedrete che soltanto javacard é presente.
18) Cliccare su Compile; guardando nella cartella javacard troveremo i files iso_639_4.isu e basic_.clu
che sono in effetti i files compilati.
19) Proviamo ora a modificare iso_639 cosi' :

IF ((Numb = 6579573)) THEN
Lig := "TEDESCO DI GERMANIA";
ELSE

20) salvare e provare a compilare; prima di compilare cliccare su Debug per vedere eventuali messaggi d'errore.

Per modificare e aprire un file .pan con HPanel, bisogna prima di tutto decompilare il file stesso con PanParser.
Dopodiché si prendono i file che sono stati creati (un file .pan, che è il nostro file decompilato, e uno .mod) e si mettono nella cartella del progetto di HPanel. Fatto ciò aprire il file .pan decompilato con l'editor di pannelli di Hpanel. Vedrete la struttura del pannello senza i veri colori. Caricate il file mod e i colori saranno quelli originali (questo file si chiama modulo e viene caricato premendo il pulsante che sembra un telefono con un antenna).

PCBootDirect
Programma anche questo ufficiale della casa creatrice del Pantalk. Molto più user-friendly rispetto ad Hpanel, e freeware, anche se con alcune limitazioni (ad esempio non vi è la possibilità di creare nuove API).
Con il programma è compresa una guida al programma che non comprende nessun tipo di configurazione particolare (tipico programma in stile Windows) a differenza di HPanel. Anche se pure qui ci sono da fare le modifiche per il file di emb4p.prt

Ora che avete imparato a usare HPanel così alla carlona, provate a fare qualche altra modifica se vi va, vi renderete presto conto che bisogna incaxxarsi diecimila volte prima di riuscire a fare qualcosa... praticamente ciò che fate con l'accoppiata ucomp+Defiant in dieci minuti, con hpanel ci mettete circa mezz'ora in più!
Per questo motivo ho deciso di integrare in parte anche HPanel con defiant!!!!!
Eh si... provando infatti ad aprire uno script con defiant avendo prima selezionato da menu Configurazione->Compilatore *.cpi la voce Usa PanParser, e solo dopo aver configurato le directory nel menu Tools->settings.
WorkDirectory è la cartella dove Defiant andrà a posizionare i file che dovrete compilare, l'altra è dove si trova l'eseguibile di Hpanel (ovviamente Hpanel/PCboot dovrà essere configurato correttamente affinchè possa compilare i file!).
Aprite ora lo script e vi si aprirà una finestra con lo script decompilato in puro pantalk, modificatelo a vostro piacimento e poi cliccate sulla freccia verde (esporta in HPanel). A questo punto andate nel menu tools e cliccate su Hpanel, vi aprirà il programma dove andrete a selezioanre il vostro progetto e i file di script da compilare. Compilate, chiudete HPanel (potete anche lasciarlo aperto se vi va) e in defiant cliccate ora sulla freccia rossa nell'editor dello script (importa da HPanel). A questo punto avrete il vostro bello script compilato, cliccando su OK vi verranno salvati i cambiamenti, se fate cancella non vi salverà nulla.
Molto più semplice a mio avviso di fare tutto manualmente... questo per ora è stato il massimo grado di automatizzazione che sono riuscito ad ideare per interfacciare HPanel con defiant... altro non mi è venuto in mente.

Come potete notare utilizzando l'editor degli script di defiant avrete maggiore controllo sullo script stesso, questo perchè l'editor di Hpanel è leggermente penoso!!

Pregi di HPanel:
- ha il compilatore che funziona bene (essendo quello ufficiale)
- panparser decompila sempre in modo corretto a differenza di ucomp, anche se alcuni script non li decompila!
Difetti:
- tutto il resto!!

Pregi di Defiant:
- avete la possibilità di mantenere sotto controllo tutto ciò che vi serve in modo semplice e veloce
- avete strumenti affidabili per la modifica e molto intuitivi.
- compilatore interno integrato che raggiunge livelli di stabilità incredibili
Difetti:
- boh! :D

A voi l'ardua sentenza sul giudicare i metodi di lavoro, io credo di averveli esposti più o meno tutti quanti.
Ognuno faccia come crede meglio e soprattutto come si trova meglio.
Conosco gente che preferisce lavorare con il linguaggio pantalk puro, altri con il simil-c di ucomp.
Conosco gente che preferisce utilizzare defiant unito a ucomp e altra che preferisce usare editplus+ucomp, usando defiant solo per gli altri file e l'estrazione dal firmware.

Fate vobis...

Pack/Unpack: la compressione dei file

Abbiamo il nostro bel firmware, siamo riusciti a estrarre tutti i file MHW ma ancora non sappiamo come interpretare questi file e come sono costituiti!

Ebbene, cercheremo in questo thread di spiegare come si svolge l'algoritmo di compressione/decompressione dei file MHW.

Come già detto, prima di inserire (compattare) i file nel firmware questi vengono compressi per poter ottenere uno spazio di occupazione dei singoli file minore rispetto a quello originale. In questo modo potremo avere molti file in poco spazio!

Vediamo come si svolge questo algoritmo che viene applicato su tutti i file eccetto quelli con estensione *.fnu (file immagini compresse in MPEG, reale estensione *.m2v):
questo algoritmo diminuisce le ridondanze dei bit all'interno del file in base a delle sequenze.
Per fare questo si inseriscono alcune informazioni preliminari che ci indicano se i byte che dobbiamo estrarre fanno parte di una sequenza già incontrata oppure no.
Nel caso la sequenza fosse già stata incontrata ci viene indicata la posizione della sequenza e quanti byte dobbiamo leggere/considerare.
Ovviamente tutte queste informazioni vengono gestite da singoli bit per poter risparmiare spazio quindi noi dovremo considerare il file come una sequenza di bit e non di singoli byte.
Avremo così delle sorte di record formate da bit con un header e il corpo dati.
Nell'header abbiamo il primo bit che identifica il tipo di record in cui ci troviamo (se vale 1 il record appartiene a una sequenza di byte che non è stata compressa e i successivi 3 bit indicano da quanti byte è costituita la sequenza stessa, per un valore massimo di 8; la base di partenza è 0 considerata come se valesse 1; se il valore del primo bit del record vale 0 allora è una sequenza già incontrata, quindi compressa, i successivi 3 bit ci indicheranno di quanti bit sarà lungo il prossimo campo del record, dove tale campo ci indica di quanti byte dobbiamo tornare indietro per poter reincontrare la sequenza di byte desiderati, subito dopo si trovano altri 2 bit che indicano quanti byte dobbiamo andare a leggere dalla precedente sequenza).

Dopo l'header si trovano i bit che costituiscono la sequenza e una volta descritti tutti i bit che vanno a formare i byte della sequenza si ha un nuovo record.
Il file compresso una volta conclusasi la compressione deve terminare con la sequenza di bit 0 001 00000. Tale informazione sarebbe il terminatore: è in modalità Distance e indica di avere bisogno di cinque bit per indicare che il BACK OFFSET è solo di uno (informazione che può e deve essere descritta con solo 4 bit invece di cinque).

Di seguito si trova l'algoritmo di decompressione così come è stato esposto da Tomas Vlad (colui che ha scoperto per primo come funzionava tale algoritmo):

```
/* unpackMHW.cpp -- demonstration of using MediaHW unpacking algorithm
  version 1.02, September 21th, 2001

  Copyright © 2001 Tomas Vlad

  This software is provided 'as-is', without any express or implied
  warranty.  In no event will the author be held liable for any damages
  arising from the use of this software.

  Permission is granted to anyone to use this software for any purpose,
  including commercial applications, and to alter it and redistribute it
  freely, subject to the following restrictions:

  1. The origin of this software must not be misrepresented; you must not
     claim that you wrote the original software. If you use this software
     in a product, an acknowledgment in the product documentation would be
     appreciated but is not required.
  2. Altered source versions must be plainly marked as such, and must not be
     misrepresented as being the original software.
  3. This notice may not be removed or altered from any source distribution.

  Tomas Vlad
  admin@hm.ukrtel.net

*/

#define PROGRAM_NAME "UnpackMHW"
#define VERSION_NUMBER "1.02"
#define VERSION_DATE "21.09.2001"

#include <stdlib.h>
#include <stdio.h>
#include <malloc.h>
#include <string.h>
#include <memory.h>
#include <direct.h>

#define END_Z (-1)
#define LITERAL 1
```

```
#define DISTANCE 0

/* And'ing with mask[n] masks the lower n bits */
unsigned mask[33] = {
   0x00000000,
   0x00000001, 0x00000003, 0x00000007, 0x0000000f,
0x0000001f, 0x0000003f, 0x0000007f, 0x000000ff,
   0x000001ff, 0x000003ff, 0x000007ff, 0x00000fff,
0x00001fff, 0x00003fff, 0x00007fff, 0x0000ffff,
   0x0001ffff, 0x0003ffff, 0x0007ffff, 0x000fffff,
0x001fffff, 0x003fffff, 0x007fffff, 0x00ffffff,
   0x001fffff, 0x003fffff, 0x007fffff, 0x00ffffff,
0x1fffffff, 0x3fffffff, 0x7fffffff, 0xffffffff
};

char *dict_buf, *z_buf;
char in_file[40];
char out_file[40];

void infl(char *name, char *zdata, int zlen)
{
char *dict, *z_now, *z_end;
int d_cnt, data, bit_pos, need_bits;
int need_off, offset, dict_len, dict_abs;
int mode, dict_now, tmp;
int i, cnt;
FILE *out;

out=fopen(name,"wb");
if(out == NULL) { printf("can't open file %s\n",name); return; }

memset(dict_buf,0,0x840);
d_cnt = 0;
dict = dict_buf;
dict_now = 1;
data = 0;
z_now = zdata;
z_end = zdata + zlen - 1;

// get mode = 1 bit
data = *z_now++;
bit_pos = 7;
mode = (data >> bit_pos) & 1;

// loop to end_data OR end_z_mode
while((z_now <= z_end) && (mode != END_Z)) {

if(mode == LITERAL) { // chars gets from input_data
```

```
// get count = 3 bit
if(bit_pos > 2) { bit_pos -= 3; need_bits = (data >> bit_pos) & 7;}
else {
need_bits = (3 - bit_pos);
tmp = ((data & mask[bit_pos]) << need_bits) & 0xff;
data = *z_now++;
bit_pos = (8 - need_bits);
need_bits = ((data >> bit_pos) & mask[need_bits]) | tmp;
}

// copy to dict & out_stream, count = cnt
cnt = need_bits + 1;
i = 0;
while(i < cnt) {
if(bit_pos > 7) {bit_pos -= 8; need_bits = (data >> bit_pos) & 0xff;}
else {
need_bits = (8 - bit_pos);
tmp = ((data & mask[bit_pos]) << need_bits) & 0xff;
data = *z_now++;
bit_pos = (8 - need_bits);
need_bits = ((data >> bit_pos) & mask[need_bits]) | tmp;
}

*(dict + dict_now) = need_bits;
fputc(need_bits,out);
d_cnt++;
dict_now = (dict_now + 1) & 0x7ff;
i++;
}
// end copy
if(cnt != 8) mode = DISTANCE;
} // end mode = LITERAL

if(mode == DISTANCE) { // chars gets from dictionary_data
// get dict_offset_size = 3 bits
if(bit_pos > 2) {bit_pos-=3; need_bits = (data >> bit_pos) & 7;}
else {
need_bits = (3 - bit_pos);
tmp = ((data & mask[bit_pos]) << need_bits) & 0xff;
data = *z_now++;
bit_pos = (8 - need_bits);
need_bits = ((data >> bit_pos) & mask[need_bits]) | tmp;
}

need_off = need_bits + 4; // dict_offset_size = (4 - 11) bits
offset = 1;
// get 8 high bits dict_offset
if(need_off > 8) {
```

```
if(bit_pos > 7) {bit_pos -= 8; need_bits = (data >> bit_pos) & 0xff;}
else {
need_bits = (8 - bit_pos);
tmp = ((data & mask[bit_pos]) << need_bits) & 0xff;
data = *z_now++;
bit_pos = (8 - need_bits);
need_bits = ((data >> bit_pos) & mask[need_bits]) | tmp;
}
offset += (need_bits << (need_off - 8));
need_off -= 8;
}
// get low bits dict_offset
if(need_off > bit_pos) {
need_bits = (need_off - bit_pos);
tmp = ((data & mask[bit_pos]) << need_bits) & 0xff;
data = *z_now++;
bit_pos = (8 - need_bits);
need_bits = ((data >> bit_pos) & mask[need_bits]) | tmp;
}
else {bit_pos -= need_off; need_bits = (data >> bit_pos) & mask[need_off];}

offset += need_bits; // back_offset = (1 <-> 2048)
dict_abs = (dict_now - offset) & 0x7ff; // dict absolute position

// test end unpacking !!!
if((offset == 1) && (need_off > 4)) {
mode = END_Z;
goto end_dcd;
}

// get len_dict_size = 2 bits
if(bit_pos > 1) {bit_pos -= 2; need_bits = (data >> bit_pos) & 3;}
else {
need_bits = (2 - bit_pos);
tmp = ((data & mask[bit_pos]) << need_bits) & 0xff;
data = *z_now++;
bit_pos = (8 - need_bits);
need_bits = ((data >> bit_pos) & mask[need_bits]) | tmp;
}

dict_len = need_bits + 3; // len_dict_cnt_size = (3 <-> 6) bits
// get copy_from_dict_cnt
if(dict_len > bit_pos) {
need_bits = (dict_len - bit_pos);
tmp = ((data & mask[bit_pos]) << need_bits) & 0xff;
data = *z_now++;
bit_pos = (8 - need_bits);
need_bits = ((data >> bit_pos) & mask[need_bits]) | tmp;
```

```
}
else { bit_pos -= dict_len; need_bits = (data >> bit_pos) & mask[dict_len];}

// copy from dict[dict_abs] to out_stream & dict, count = cnt
cnt = need_bits + 1; // loop_count = len_idx + 1 = (1 <-> 64)
i = 0;
while(i < cnt) {
need_bits = *(dict + ((i + dict_abs) & 0x7ff));
*(dict + dict_now) = need_bits;
fputc(need_bits,out);
d_cnt++;
dict_now = (dict_now + 1) & 0x7ff;
i++;
}

} // end mode = DISTANCE

// get mode = 1 bit
if(bit_pos > 0) { bit_pos-=1; mode = (data >> bit_pos) & 1; }
else { data = *z_now++; bit_pos = 7; mode = (data >> bit_pos) & 1; }

} // end while

end_dcd:

fclose(out);
// unpacking info
printf("z_data_len = %08X , decode_len = %08X\n", zlen, d_cnt);
printf("remain data len = %08X\n", z_end - z_now + 1);
printf("stop: data = %02X, bit_pos = %d\n", data & 0xff, bit_pos);

return;
}

void usage(void)
{
printf("\n%s  --  version %s of %s.\n", PROGRAM_NAME, VERSION_NUMBER,
VERSION_DATE);
   printf("Usage: %s in_file [out_file]\n",PROGRAM_NAME );
   printf("       This unpack MHW in_file to out_file.\n");
   printf("©TomasVlad MediaHW unpacker admin@hm.ukrtel.net\n");
   exit(0);
}

void getparams(int argc,char *argv[])
{
   char *c;
```

```
   in_file[0] = '\0';
out_file[0] = '\0';

   ++argv;
   if(argc < 2) { usage(); exit(0); }

if(--argc > 0) { c = *argv; strcpy(in_file, c); }
++argv;
if(--argc > 0) { c = *argv; strcpy(out_file, c); }

if(out_file[0] == 0) strcpy(out_file,"out.mhw");

}

int main(int argc, char* argv[])
{
int i, c;
FILE *fin;

z_buf = (char*)malloc(0x1000);
dict_buf = (char*)malloc(0x840);

if((z_buf == NULL) || (dict_buf == NULL)) {
printf("Can't get memory\n");
exit(1);
}

getparams(argc, argv);

   fin = fopen(in_file, "rb");
   if (fin == NULL) {
printf("File %s not found\n", in_file);
exit(1);
   }

memset(z_buf,0,0x1000);

i=0;
while((c=fgetc(fin)) != EOF) z_buf[i++] = c;

fclose(fin);

if(i == 0) {
printf("\nInput file len == 0\n");
exit(1);
}

infl(out_file,z_buf,i);
```

```
return 0;
}
```

Qui invece ho esposto l'attuale algoritmo di compressione/decompressione utilizzato in Defiant, è l'algo che ha utilizzato anche pgete ma ho modificato alcuni controlli e altre piccole cose per cercare di migliorare l'algoritmo di compressione riuscendo a migliorarlo un pochetto (anche un singolo bit a volte è prezioso perchè se considerato su grandi quantità diventano byte, poi kilobyte, ecc...)

```
unit Pack;

// ---------------------------------------------- //
// Esta unidad es una version en Delphi del código //
// del programa Pack/Unpack de Pgete modified by Tideglo //
// ---------------------------------------------- //

interface

function UnPackMHW(origen, destino : PChar; longitud : integer):integer;
function PackMHW(origen, destino : PChar; longitud : integer; modo : byte):integer;

implementation

// lee siguiente bit
function RdBit(buftemp : PChar; var posicion : integer):byte;
begin
  RdBit := ((ord(buftemp[posicion shr 3]) shr (7-(posicion mod 8))) and $01);
  inc(posicion);
end;

// escribe siguiente bit
procedure WrBit(buftemp : PChar; var posicion : integer; valor : boolean);
begin
  if valor then
    buftemp[posicion shr 3] := chr(ord(buftemp[posicion shr 3]) or ($80 shr (posicion mod 8))) //
SET
   else
    buftemp[posicion shr 3] := chr(ord(buftemp[posicion shr 3]) and not($80 shr (posicion mod 8)));
// CLR
  inc(posicion);
end;

// AnchoLongitud=2bits+3
procedure AnchoLongitud(longitud : integer; destino : PChar; var PosBitOut : integer);
var
  i : integer;
```

```
begin
  i := 0;
  while ((1 shl (i+3))<longitud) do
    inc(i);
  WrBit(destino, PosBitOut, ((i and $2)>0));
  WrBit(destino, PosBitOut, ((i and $1)>0));
  i := i + 3;
  while (i>0) do
    begin
      WrBit(destino, PosBitOut, (((longitud-1)and(1 shl (i-1)))>0));
      dec(i);
    end;
end;

// AnchoOffset=3bits+4
procedure AnchoOffset(offset : integer; destino : PChar; var PosBitOut : integer);
var
  i : integer;
begin
  i := 0;
  while ((1 shl (i+4))<offset) do
    inc(i);
  WrBit(destino, PosBitOut, ((i and $4)>0));
  WrBit(destino, PosBitOut, ((i and $2)>0));
  WrBit(destino, PosBitOut, ((i and $1)>0));
  i := i + 4;
  while (i>0) do
    begin
      WrBit(destino, PosBitOut, (((offset-1)and(1 shl (i-1)))>0));
      dec(i);
    end;
end;

// almacenar X caracteres en modo comprimido (se suponone que ya estamos en ese modo)
procedure Write_Comprimido(Offset, longitud : integer; var PosInp : integer;
                           var count : integer; destino : PChar; var PosBitOut : integer);
begin
  AnchoOffset(offset, destino, PosBitOut);
  AnchoLongitud(longitud, destino, PosBitOut);
  PosInp := PosInp + longitud;
  count := 0;
end;

// almacenar X caracteres en modo no comprimido
procedure Write_NoComprimido(caracteres : integer; var PosInp : integer; var PosBitInp : integer;
                             var count : integer; origen, destino : PChar; var PosBitOut : integer);
var
  i : integer;
```

```
begin
  i := caracteres shl 3;
  WrBit(destino, PosBitOut, True);
  WrBit(destino, PosBitOut, (((caracteres-1) and $4)>0));
  WrBit(destino, PosBitOut, (((caracteres-1) and $2)>0));
  WrBit(destino, PosBitOut, (((caracteres-1) and $1)>0));
PosBitInp := PosInp shl 3;
while (i>0) do
    begin
      WrBit(destino, PosBitOut, (RdBit(origen, PosBitInp)=1));
      dec(i);
    end;
  PosInp := PosInp + caracteres;
end;

// añade los bits necesarios para que le fichero sea byte-alineado
procedure Alinear(destino : PChar; var PosBitOut : integer);
begin
  while ((PosBitOut mod 8)>0) do
    WrBit(destino, PosBitOut, False);
end;

function UnPackMHW(origen, destino : PChar; longitud : integer):integer;
var
  PosBitInp, PosBitOut, LongFileBits, offset, length, width, i : integer;
begin
  LongFileBits := longitud shl 3;    // contendra bits en el fichero de entrada
  PosBitInp := 0;
  PosBitOut := 0;                    // Unicas variables que deben inicializarse
  while (LongFileBits>PosBitInp) do  // Prevenir fichero vacío
    begin
      length := 8;
      while (length=8) and (RdBit(origen, PosBitInp)=1) do  // modo SinComprimir
        begin
  length := ((RdBit(origen, PosBitInp) shl 2) or
                  (RdBit(origen, PosBitInp) shl 1) or
                  (RdBit(origen, PosBitInp))) + 1;  // Longitud de bytes a copiar
  i := length shl 3;                        // bits a copiar
         while (i>0) do
           begin
             WrBit(destino, PosBitOut, (RdBit(origen, PosBitInp)=1));
             dec(i);
           end;
        end;
      length := 0;                              // modo Comprimido
      offset := 0;
      Width := ((RdBit(origen, PosBitInp) shl 2) or
              (RdBit(origen, PosBitInp) shl 1) or
```

```
            (RdBit(origen, PosBitInp))) + 4;       // ancho de Offset
      while (Width > 0) do
       begin                        // Offset a bytes anteriores en out (+1)
         offset := offset shl 1;
         offset := offset or RdBit(origen, PosBitInp);
         dec(width);
       end;
      Width := ((RdBit(origen, PosBitInp) shl 1) or
            (RdBit(origen, PosBitInp))) + 3;       // ancho de Length
      while (Width > 0) do   // Longitud/Cuenta de bytes a copiar desde out (+1)
       begin
         length := length shl 1;
         length := length or RdBit(origen, PosBitInp);
         dec(width);
       end;
      if (offset > 0) then              // Repetición cadena anterior
       begin
         if ((PosBitOut shr 3)-(offset+1))>=0 then
           for i := 0 to length do
             destino[(PosBitOut shr 3)+i] := destino[(PosBitOut shr 3)-(offset+1)+i];
         PosBitOut := PosBitOut + ((Length+1) shl 3);
       end
      else
       if (length > 0) then
         begin
           if ((PosBitOut shr 3)-(offset+1))>=0 then
             for i := 0 to length do     // Repetición del último carácter
               destino[(PosBitOut shr 3)+i] := destino[(PosBitOut shr 3)-(offset+1)];
           PosBitOut := PosBitOut + ((Length+1) shl 3);
         end;
    end;
  Alinear(destino, PosBitOut); // Hay un caso de fichero 10 00 empaquetado sauve.!?
  UnPackMHW := PosBitOut shr 3;
end;

function PackMHW(origen, destino : PChar; longitud : integer; modo : byte):integer;
var
  PosInp, count, PosBitOut, PosBitInp : integer;
  i, j, l, LengthMax, OffsetMax, rep : integer;
  w : byte;
begin
  if ((modo and $4)>0) then     // Modo and Usa_9
    w := 9                    // opcion 9 mas compresión no usada en original
   else
    w := 8;
  PosInp := 0;                 // posición absoluta byte en cadena INP
  count := 0;                 // posición relativa actual en estudio respecto a PosInp
  PosBitInp := 0;              // posición absoluta bit en cadena INP (se recalcula con PosInp)
```

```
  PosBitOut := 0;              // posición absoluta bit en cadena OUT
  while (longitud>0) do        // por si fichero de longitud 0
   begin
    // 1º ------ comprobar si Final de fichero  --------
    if (PosInp + count >= longitud) then      // > no debería ser nunca
     begin
      if (count>0) then
       Write_NoComprimido(count, PosInp, PosBitInp, count, origen, destino, PosBitOut); //
almacenar count caracteres
      if (count=8) or (count=0) then        // cambia a modo comprimido
       WrBit(destino, PosBitOut, False);   // .. en estudio finales del fichero
      //AnchoOffset(1, destino, PosBitOut);   // bits
      //AnchoLongitud(1, destino, PosBitOut); // indicando final
      WrBit(destino, PosBitOut, False);
      WrBit(destino, PosBitOut, False);
      WrBit(destino, PosBitOut, True);
      for i:=1 to 5 do
       WrBit(destino, PosBitOut, False);
      Alinear(destino, PosBitOut);         // alinear a bytes
      Break;                               // y terminar
     end;
    // 2º -- siguiente a estudiar (aún no almacenado) --
    inc(count);
    // 3º ----------- Grupos 8 caracteres ---------------
    if (count = w) then
     begin                                 // damos oportunidad aún a que los próximos
      Write_NoComprimido(8, PosInp, PosBitInp, count, origen, destino, PosBitOut); // ..
caracteres enlacen al 8º, el original
      count := 0;                          // .. parece no usarlo y pierde algunos bytes ;)
      continue;                            // tras cualquier operación volver a chequear estado en fichero
     end;
    // 4º ----------- Cadenas repetidas -----------------
    if ((modo and $1)>0) then              // Modo and Usa_Cadenas
     begin
      LengthMax := 0;
      OffsetMax := 0;
      if ((PosInp+count-1)<2048) then      // ajustar a máximo offset
       i := 0
      else
       i := PosInp+count-1-2048;
      while (i<(PosInp+count-1)) do        // caracter inicial a buscar hasta el anterior en estudio
       begin
        if (origen[i]=origen[PosInp+count-1]) then   // si coindide estudiar longitud de la cadena
equivalente 1...
         begin
          j := i+1;
          l := 1;
          while (*(j<(PosInp+count-1)) and *)         // creo que el original pone
```

```
for(j=i+1;j<LongFile;j++)
                                        // .. desaprovechando bytes en ocasiones y no permitiendo
memcpy en upack!
                  (origen[j] = origen[(PosInp+count-1)+l]) do  // si no coincide buscar otra posible
cadena desde otro inicio
                begin
                  inc(l);
                  if (l>=LengthMax) then              // >= para quedarse con el offset menor (posibles
menos bits)
                    begin
                      LengthMax := l;
                      OffsetMax := (PosInp+count-1)-i;
                    end;
                  if ((l=64) or ((PosInp+count+l)>=longitud)) then
                    break;                          // no mas de 64 ni fuera de fichero
                  inc(j);
                end;
              end;
            inc(i);
          end;
        if (LengthMax>2) then                       // >1 en el original con lo que
          begin                                     // .. para almacenar una cadena de 1 byte ocupan más !!!
            dec(count);                             // el actual ya pertenece a la cadena y no se almacena
            if (count>0) then                       // almacenar count anteriores NO INCLUYE ACTUAL
              Write_NoComprimido(count, PosInp, PosBitInp, count, origen, destino, PosBitOut);
            if (count=8) or (count=0) then          // si no escribimos nada antes (8 no posible aquí)
              WrBit(destino, PosBitOut, False);     // cambia a modo comprimido
            Write_Comprimido(OffsetMax, LengthMax, PosInp, count, destino, PosBitOut);
            continue;                               // tras cualquier operación volver a chequear estado en
fichero
          end;
      end;
    // 5º ---------- Caracteres repetidos----------
    if ((modo and $2)>0) then       // Modo and Usa_Repetidos
      begin
        rep := 0;
        while (origen[PosInp+count-1] = origen[PosInp+count+rep]) and
            ((PosInp+count+rep)<longitud) and (rep<64) do
          inc(rep);
        if (PosInp>0) and (rep>0) and (count=1) and (origen[PosInp-1]=origen[PosInp]) then
          begin
            if rep<>64 then inc(rep);
            count := 0;             // si el anterior caracter almacenado por una cadena coincide aumentar
rep y ajustar count
          end;
        if (rep>1) then             // rep=1 no es posible porque lo usamos para acabar el programa
          begin
            if (count>0) then
```

```
            Write_NoComprimido(count, PosInp, PosBitInp, count, origen, destino, PosBitOut);   //
almacenar count anteriores DEBE DE INCLUIR CHAR ORIGINAL!!!
           if (count=8) or (count=0) then         // si escribimos 8
            WrBit(destino, PosBitOut, False);    // cambia a modo comprimido
           Write_Comprimido(1, rep, PosInp, count, destino, PosBitOut);
           continue;
          end;
       end;
    end;
  PackMHW := PosBitOut shr 3;         // Tamaño del fichero a grabar
end;

end.
```

Glue/GlueDump

Questo programma è utile per estrarre i file MHW dal firmware. E' stato fatto da mat e una parte di codice da Virg0s.

Italian OpenFirmware Project

Qualche anno fa (non ricordo precisamente quando) ci fu sk4s4t che all'epoca faceva parte dell'FSWT (Firmware Sky Walker Team, un forum dedicato ai decoder e allo sviluppo dei firmware, uno dei più in voga assieme al Sif Team, Strong Italian Forum Team, e al DDTKForum, Digital Dream Team Klan Forum).
Sk4s4t nel 2003 decise di iniziare un progetto carino, creare un firmware mhw partendo da zero, così creò il suo spazio su sourceforge.net e molti grandi personaggi italiani che si dedicavano alla modifica dei firmware si riunirono.
Purtroppo non ci fu mai una collaborazione definitiva, non conosco i motivi ma sk4s4t cercò il più possibile di portarlo avanti e grazie ad LvX ci fu la nascita del primo firmware nato da questo progetto, successivamente il progetto fu abbandonato principalmente perchè nessuno collaborava.

Elementi base del linguaggio:

Gli operatori

In questo paragrafo farò un elenco degli operatori utilizzati dal linguaggio.

Operatori Aritmetici

Simil-C:

"=" assegnazione -> var= 1+4;
"*" Moltiplicazione -> var= 1*4;
"/" Divisione -> var= 1/4;

"-" Sottrazione -> var= 1-4;
"+" Addizione -> var= 1+4;
"%" Modulo -> var= 9%2;
Pantalk puro:
":=" assegnazione -> var:= 1+4;
"*" Moltiplicazione -> var:= 1*4;
"/" Divisione -> var:= 1/4;
"^" Potenza -> var:= 1^4;
"-" Sottrazione -> var:= 1-4;
"+" Addizione -> var:= 1+4;
"%" Modulo -> var:= 9%2;
In entrambi le tipologie del linguaggio esiste anche l'operatore "-" utilizzato per la indicare un numero negativo, esempio var=-1;

Operatori di Comparazione (tanto per intenderci quelli usati nelle condizioni)

Simil-C:
"<=" minore uguale
">=" maggiore uguale
"!=" diverso
"<" minore
">" maggiore
"==" uguale
Pantalk puro:
"<=" minore uguale
">=" maggiore uguale
"<>" diverso
"<" minore
">" maggiore
"=" uguale

Operatori logici

Simil-C
"&&" AND logico -> if ((var <= 5) && (var1<>1)){
"||" OR logico -> if ((var >= 5) || (var1<>0)){
"!" NOT logico -> if (!(var >= 5) || !(Var1<>0)){
Pantalk puro:
":AND:" AND logico -> if (var <= 5) :AND: (var1<>1) then
":OR:" OR logico -> if (var >= 5):OR: (var1<>0) then
"!" NOT logico -> if !(var >= 5) :OR: !(Var1<>0) then

Operatori relativi ai bit

Simil-C:
"&" AND bit per bit
"|" OR bit per bit
"^" OR exclusive bit per bit

"<<" Shift a sinistra
">>" Shift a destra
"!" NOT bit per bit
Pantalk puro:
":BAND:" AND bit per bit
":BOR:" OR bit per bit
":BXOR:" OR exclusive bit per bit
":BSL:" Shift a sinistra
":BSR:" Shift a destra
"!" NOT bit per bit

Per fare riferimento agli array (vettori) si usa in entrambe le tipologie di linguaggio l'operatore "[indice]".

Le parentesi "()" impostano l'ordine di esecuzione delle operazioni nell'assegnazione o di interpretazione delle condizioni.

La separazione degli elementi nelle chiamate alle funzioni avviene tramite ",".

La concatenazione delle stringhe avviene tramite l'operatore "+". In entrambi i linguaggi le stringhe costanti si scrivono tra i doppi apici ' " ' (esempio: "Mi piacciono le tettone!")

Per fare riferimento alla variabile di una classe si usa in entrambi i linguaggi classe.variabile. Nel caso la classe sia quella dichiarata di default si può omettere il nome della classe usando semplicemente il nome della variabile.

I commenti vanno inseriti in simil-C usando questa coppia di simboli "//", da qui in poi tutta la riga è commentata, oppure usando “/*” tutto quello qui in mezzo è commentato “*/”. In pantalk puro si usa l'accoppiata di simboli "/*" e "*/", tutto ciò che si trova tra questi simboli è da considerarsi commento.

Nel simil-C esistono anche tutti gli operatori di assegnazione come "+=", "-=", "|=", "&=", "^=", "var++", "var--".

Nel simil-C per indicare il puntamento alle variabili si usano "*" e "&" allo stesso identico modo del C.

Per assegnare valori si utilizza la notazione decimale (10) oppure quella esadecimale (0xA).

Alle variabili vengono assegnati valori iniziali come da tabella qui sotto

Integer	0
Real	0
Text	""
Alphanu	""
Alpha	""
Format	0:00

Formattazione delle Variabili

Decompilando alcuni script vi potrebbe capitare di incontrare una specie di stringa ma con una struttura un pò particolare e passata come parametro per qualche funzione (spesso nelle funzioni Copy_Mem_To_Var e Copy_Var_To_Mem).

Ebbene queste "stringhe" sono in realtà delle formattazioni di variabili!

Vediamone un esempio:

"1\nperiod:2\nduree:14\ntype:8\nopi:16"

Ma come va interpretato?
Semplice, o per lo meno lo dovrebbe essere!

La struttura è questa qui.
"**Block_Size**\nIdentifier1**:**Block_Count1\nIdentifier2**:**Block_Count2..."

- Block_Size: è il parametro che indica di quanti bit è formato ogni blocco, può avere valore 1 o 8
- Identifier1, Identifier2: è l'identificatore del campo e può essere il nome di una variabile, una costante o un "-". Ogni costante deve essere preceduta da uno "0" davanti (questo perchè i valori costanti sono espressi in esadecimale) e può avere dimensione massima di 32 bits. Il "-" indica uno spostamento nel buffer associato.
- Block_Count1, Block_Count2: è il parametro che indica quanti blocchi prendere in considerazione per l'attuale campo preso in considerazione. Nel caso block_size sia 1 e block_count 2, si prenderanno 2 bit; se fosse stato blocksize=8 e blockcount=3 si prendevano 3 byte.
- \n: indica l'inizio di una nuova riga, ossia di un nuovo campo. Il primo campo è sempre il block_size, gli altri sono tutti della struttura identifier:block_count.

Strutture di controllo

Partiamo ora ad analizzare il linguaggio Pantalk per quello che è realmente, ossia il suo codice e le sue funzionalità.

Ci terrei a sottolineare due aspetti fondamentali... il primo è che essendo il pantalk molto dinamico è logico che può essere variato a piacimento, il secondo è che esistono principalmente due tipi di linguaggio, il Pantalk puro e quello Simil-C.

Noi analizzeremo principalmente il pantalk Simil-C (quello utilizzato dal compilatore interno a Defiant e da uComp anche se con qualche differenza tra i due di sintassi), questo perchè mi sembra molto più bello da vedere e probabilmente in molti hanno più familiarità con l'utilizzo del C piuttosto che di un linguaggio miscugliato tra pascal (non molto diffuso) e il visual basic.

Ad ogni modo cercherò, quando mi è possibile, di mostrare le differenze tra i due (almeno nelle strutture di controllo, visto che le chiamate alle api possono essere variate tranquillamente senza dipendere dal linguaggio), facendo un esempio con ogni tipo di linguaggio (questa mi pare essere l'unica soluzione per impararli entrambi o per chi vuole solo uno dei due...). Come sempre potrei dire castronerie e io ripeto che non sono un maestro e non ho intenzione di spiegare le basi della

programmazione. Quindi... o sapete programmare o vi attaccate al... tram!

Ultima cosa e poi iniziamo, le chiamate alle API del pantalk non sono mai state definite con uno standard, diciamo che ognuno se vuole le può chiamare come vuole.
Tanto per farvi un esempio, se noi facciamo riferimento alla funzione API numero 0x12C, questa può essere chiamata Unload_Module oppure UnloadModule o ancora ModuleUnload ecc... in poche parole il riferimento mnemonico può essere variato a piacimento, ma quel che conta è che sappiamo che facendo riferimento a quel nome noi andremo a richiamare la proc_012C che compie una certa azione. Per questi motivi quando andrò a descrivere le api le chiamerò prima con il riferimento al numero della api stessa e successivamente vedremo il nome mnemonico.

Sia in HPanel che in Defiant la decompilazione/compilazione può essere variata nei suoi mnemonics, ma in uComp no! Questo perchè pGete ha tutto parametrizzato via codice e ha fatto delle sue scelte personali che a volte sono condivisibili mentre altre un pò meno, vedrete come alcune funzioni abbiano nomi piuttosto poco immediati in uComp.

Detto ciò iniziamo.

La struttura condizionale **if else**.

Simil-C

```
if(condizione){
  codice
}

if(condizione){
  codice
}
else{
  codice
}
```

Pantalk puro:

```
IF (condizione) THEN
 codice
ENDIF

IF (condizione) THEN
 codice
ELSE
 codice
ENDIF
```

Semplice? Semplicissimo!
Notate come il simil-C sia uguale al C, come vedremo possiamo notare che tutto il simil-c nelle strutture di controllo è uguale al C.

Ricordiamoci di una cosa. Il C è case-sensitive, quindi anche uComp (basandosi su un compilatore di c) è case-sensitive mentre il compilatore interno a Defiant non è case sensitive per permettere una facilità di utilizzo maggiore durante la scrittura di codice. (come il pantalk puro)
Il pantalk puro invece dovrebbe essere NON case-sensitive.

Notate, nonostante tutto, quanto sia molto elegante il pantalk puro.

Il ciclo **for**.

Simil-C:

```
for(assegnazione_iniziale ; condizione ; operazione_ciclica){
  codice
}
```

Pantalk puro:

```
FOR assegnazione_iniziale TO valore_finale ;
  codice
NEXT

FOR assegnazione_iniziale TO valore_finale STEP valore ;
  codice
NEXT
```

Notate come la struttura for del pantalk sia simile al VB con next e step.
Piccola nota:
nel for simil-c c'è una condizione che fa terminare il ciclo for, mentre nel pantalk puro si esce dal for solo quando si è raggiunto e superato il valore finale (se impostate il for da 1 a 10, avrete un ciclo che verrà eseguito 10 volte).
Nel simil-C bisogna tenere presente che è possibile stabilire condizioni solo che effettuino il controllo sul raggiungimento del valore desiderato. (ossia condizioni "<=", in poche parole rendete il ciclo for uguale a quello pantalk puro)), inoltre bisogna dichiarare l'incremento della variabile mentre in pantalk puro non è necessario.
Nel pantalk puro l'incremento della variabile "dedicata" al ciclo è aumentata "automaticamente" di uno. Utilizzando STEP nel costrutto del for potremo decidere di far aumentare la variabile del valore stabilito dopo STEP ad ogni ciclo.

Il ciclo **while**.

Simil-C:

```
while(condizione){
  codice
}
```

Pantalk puro:

```
WHILE (condizione)
```

```
  codice
DONE
```

Questo ciclo a differenza del for non incrementa variabili ogni ciclo, ma esegue il codice al suo interno fino a quando la condizione impostata risulta essere vera.

Il ciclo **repeat until/do while**.

Simil-C:

```
do{
  codice
}while(!(condizione));
```

Pantalk puro:

```
REPEAT
  codice
UNTIL(condizione);
```

Questo ciclo viene eseguito fino a quando la condizione è falsa, appena risulta essere vera esce. Notate una piccola differenza. Nel Simil-C non esiste il repeat until, ma è sostituito dal do while. Notare una cosa... se provate a scrivere while(condizione) senza la negazione della condizione (ossia non scrivere while(!(condizione))) uComp e il compilatore interno a Defiant vi daranno errore perchè non lo riconoscono come un ciclo valido. Ragionando infatti possiamo creare un repeat until solo impostando la negazione nel while.
Ovviamente questa cosa è stata impostata obbligatoria perchè non esiste un codice pcode che permetta di creare un ciclo do while reale, ma solo un repeat until.

Uscite dallo script.

Cerchiamo di associare ogni script come se fosse una funzione singola in C.
Ogni script può chiamare un altro script tramite una funzione che andremo a vedere in dettaglio successivamente e che si chiama Call_Script.

Con questa funzione richiamiamo uno script e passiamo il controllo allo script chiamato. Lo script chiamante riavrà il controllo (ripartendo dalla istruzione successiva alla Call_Script, in patalk credo si richiami con SCRIPT) solo se l'esecuzione dello script chiamato è terminata.

Uno script termina quando incontra un **return, uno stop o quando non ci sono altre istruzioni da interpretare** (lo script è finito!).

Quindi...

Simil-C:

```
return;
```

Pantalk puro:
RETURN

Simil-C
break;

Pantalk puro:
STOP

PS: l'istruzione break/STOP a differenza del return e della fine dello script non fa passare il controllo allo script chiamante, ma semplicemente termina l'esecuzione del codice fino a che non succede un evento che ne reimplica l'esecuzione (non necessariamente l'esecuzione dello stesso codice). Il firmware resta per questo motivo in stallo e sconsiglio l'uso del break, normalmente lo si utilizza per fare delle prove, ma non è segno di buona programmazione se viene utilizzato nell'uso normale! (attenzione il break non ha nulla a che fare con il break che si usa in C che permette l'uscita da controlli di loop)

Dichiarazione delle classi negli script

Aprendo un file di script e decompilandolo con ucomp o con il decompilatore interno di defiant notiamo delle dichiarazioni di classi all'inizio del file.
Ebbene si, senza queste dichiarazioni ucomp e il compilatore interno a Defiant daranno errore nella compilazione perchè necessita di sapere quali classi verranno utilizzate nello script.
A dire il vero non è a ucomp e nemmeno a Defiant che interessano ma alla VM del pantalk. Per questo motivo è necessario dichiarare le classi di variabili che desideriamo utilizzare.

#Default Class nomeclasse1
#Public Class nomeclasse2
#Private Class nomeclasse3

La classe dichiarata come default (può essere soltanto una) è quella che viene presa in considerazione nel caso durante la compilazione si decida di omettere il nome della classe quando si utilizza una variabile
Esempio:
nomeclasse2.variabile1 = variabile2;

In questo caso variabile2 verrà presa come se fosse parte della classe nomeclasse1 che abbiamo dichiarato come classe di default.

Le classi dichiarate come Public sono classi che sono presenti nel firmware caricate in memoria in un qualsiasi modulo.
Le classi Private invece, sono le classi che sono presenti solo all'interno del modulo in cui si trova lo script.

Se utilizzate la decompilazione via panparser non dovrete dichiarare alcuna classe se non quella considerata di default, il tutto viene dedotto automaticamente durante la compilazione (a volte

potrebbe sbagliare...)

Credo di aver detto tutto...

Passaggio di parametri negli script

Può capitare che in alcuni casi ci serva passare alcuni valori calcolati run-time allo script che vogliamo chiamare attraverso la funzione callscript.

Molti potrebbero pensare di usare variabili appositamente dedicate allo scopo, ma come ogni linguaggio che si rispetti in patalk è possibile decidere di associare dei parametri alla chiamata di uno script (una sorta di funzione a tutti gli effetti).

Come fare?
Beh è semplicissimo, basta dichiarare nomeclasse.nomevariabile tra le parentesi che ci sono dopo il nome dello script che è presente all'inizio. Ovviamente ogni parametro è separato da una virgola ",".

Esempio:

#Script mioscript(tipovar nomeclasse.nomevariabile, tipovar1 nomeclasse1.nomevariabile1, ...)

Compilando ucomp nella decompilazione vi aggiungerà anche il tipo di parametro a cui fa riferimento la variabile. Sono informazioni solo di caraterre informativo e non a scopo di inserirle nello script, infatti se analizziamo il codice OpCode vedremo che non c'è possibilità di capire se il parametro a cui fa riferimento la variabile è di tipo intero o reale, ma è possibile farlo solo analizzando la dichiarazione della variabile all'interno della classe in cui è dichiarata la variabile stessa. Purtroppo ucomp è stato fatto in modo che sia necessario dichiarare anche il tipo di variabile, anche se nella decompilazione metterà il tipo che è stato dichiarato nella classe.

Esempio pratico:

abbiamo la variabile ciccio dichiarata nella classe festa e di tipo INTEGER
#Script festeggiato (INTEGER festa.ciccio)

Questo è il codice per avere un parametro intero da passare allo script e che sarà inserito nella variabile festa.ciccio

L'OpCode

Aprendo un file di script e decomprimendolo vedremo solamente numeri. Ovvio no? La compilazione del Pantalk avviene in OpCode che può a sua volta essere trasformato in P-Code e successivamente in Pantalk.

Ad ogni modo vi consiglio di basarvi su defiant! Già defiant, i sorgenti sono pubblici (almeno quelli della versione 2.61, gli altri no). Comunque li c'è una parte dedicata proprio alla gestione della decompilazione in simil-c. Ovviamente come tutto defiant anche questa decompilazione è parametrizzata e dinamica, grazie all'utilizzo di un file che si chiama asm_inst.ini.

Aprendolo noterete che non si capisce nulla... hahahah
Ragionandoci un pò su e basandomi pure io su questo (modificandolo un pochetto)... ho tirato fuori

un asm_inst (non ancora perfetto, è troppo lungo controllare ogni funzione!) che spiega un pò quel che fa. Almeno un pò di più si capisce, ad ogni modo vi consiglio vivamente di guardarlo con il sorgente di defiant sotto gli occhi, la parte che dovrete guardare è nel file MHWScript, seguendolo nella decompilazione interna.

Il file asm_inst è presente nella directory ini del programma di defiant.

La parte importante è l'header dove spiega un pò che cosa aspettarsi...

```
;A - top of evaluation stack, B - next value in stack, C- next ...
;primary instructions use inline parameters and push them into evaluation stack
;secondary instructions use stacked parameters (pop - number of stack cells used) and push result
into stack
;jump instructions use one integer parameter and not use stack
;functions - same as secondary, only without pushing result into stack
;oplen - length of instruction (bytes) without parameter length. If not defined then used default
value =2
;par - parameter type. May be  int, char or var (variable name)
;parlen - length of parameter (bytes)
;pop - number of values popped from stack (and used by instruction)
;push - number of values pushed into stack after completition (0 or 1, default is 0)
;modpop, modpush, shift - keys used for decompiling
;
; ATENTION!!! THIS IS NOT A REGULAR INI FILE!!!!!
; DON'T ADD ENTRIES UNLESS YOU WANT TO ADD OR MODIFY A VALUE
; DON'T PUT SPACES IN INTEGER VALUES
;
;[0000] ;hexadecimal value of the OP Code
;Name=Name of the function
;declaration=How the function is declared
;comment=A comment for the function that describes what it does.
;type=type of the function (primary, if it's a function that have a role only in OP Code; secondary,
;if it's an operation base; function, if it's a function, generally the ones from to 0x12C to 0x309;
;loop, if it's a loop like instruction; jump, if it's a goto like instruction or a conditional jump
;like if..else instruction); by default it's function
;oplen=OPCode Length, by default it's 2
;par=The type of function's parameters (int, if it's an integer value; var, if it's a class variable;
;char, if it's a string;); by default it's a null string
;parlen=Parameters length in bytes
;shift=used for the text alignment, used in if and loop instruction, you can increment or decrement
;of any value the current alignment
;peek=if not 0 then read from stack a value without delete it from stack
;modpush=used for push in the loops
;modpop=unused at the moment
;push=if not 0 it must be pushed for some next function
;pop=Number of parameters of the function to be popped
;mnem=How the function must be look, the function declaration with parameters sobstituted by
;a # delimiter; by default it's a null string
;
```

Fino alla funzione 0x12C esclusa potrete notare che nel campo Name c'è il codice in P-Code corrispondente al valore in OpCode (il valore scritto tra le parentesi quadre).

Tutto qui, altro non so come spiegarlo perchè, ripeto, credo che l'unica soluzione sia cercare di capire come decompila defiant i file, solo così avrete una idea generale del come funziona, per il resto ci sono troppe regole da esplicare scrivendole e penso che creerebbero solo più confusione e non sarei nemmeno in grado di esplicarle nel modo giusto.

Solo provando a crearsi il programma da soli e cercando di capire come viene interpretato il codice si può avere un'idea generale.

La gestione delle Api

Prima di iniziare a spiegare le varie api che vanno a costituire il pantalk, faccio una breve introduzione a come sono interpretate dal firmware queste api.

Le api non sono altro che delle funzioni scritte in un linguaggio e compilate appositamente per la macchina su cui devono essere eseguite.
Normalmente sono scritte in C, ma se uno vuole le può scrivere nel linguaggio che più gli aggrada basta che poi quando vengono compilate siano interpretabili dal processore, ossia che ci sia primo l'atto della conversione in asm macchina e poi in binario affinchè la macchina comprenda il codice. Esattamente come avviene quando si compila un programma per PC.

Per essere chiamate API devono però lavorare in un sistema operativo, questo sistema nel nostro caso è il VMOS (Virtual Machine Operative System). La VMOS (al femminile perchè la virtual machine.... non il virtual machine... ma fate come vi pare!) ha il compito di interpretare il pantalk e ogni volta che incontra una API la cerca in una tabella dove è riportato il suo indirizzo nel firmware e tutte le caratteristiche che deve avere (numero di parametri, parametri passati, ecc...)

Tutto quello che riguarda la composizione delle API, è parte dell'HDL (Hardware Dependant Layer). Come si capisce dal nome è tutta una serie di codice che non necessariamente è presente in altri STB e che non necessariamente, se esiste, deve essere uguale per tutti i modelli, appunto dipende dalla struttura hardware a cui facciamo riferimento.

Parlando in generale le API sono tutte numerate in una tabella che si chiama **ProcTable**.

La proctable è così costituita (e dovrebbe essere sempre presente in ogni firmware):

Bytes	Descrizione
0 - 1 (2 bytes)	Nome della Proc_ (al nome vanno aggiunti all'inizio i caratteri "proc_" e al numero esadecimale che si ricava da questo campo va sommato 0x12C)
2 - 3 (2 bytes)	Separatori (sempre 0x00)

4 – 7 (4 bytes)	Indirizzo dei dati della Proc_
8 (1 byte)	N° parametri della Proc_
9 - 19 (11 bytes)	Tipo di parametri

Come vedete facendo riferimento a questa tabella si trova il codice della proc e il suo indirizzo, nonchè come è stata dichiarata.

Ora che abbiamo visto come è composta la ProcTable dovremo aver capito parecchie cose sul come viene interpretato il pantalk e potremo iniziare a descrivere cosa fa ogni API.
Rimane solo una cosa da dire e che non si deve dare per scontata. Alcune funzioni (come ad esempio quella che Defiant e ucomp chiamano itos e che serve a convertire un integer in una stringa) non sono API, ma bensi funzioni facenti parte della VMOS e queste funzioni sono le funzioni BASE. Perchè base? Perchè queste sono presenti in qualsiasi sistema voi utilizziate il pantalk. Tutte le API invece (le funzioni che hanno valore dal 12C in su) non sono necessariamente presenti e possono avere significati diversi in base all'ambiente a cui si riferiscono!!

Con queste informazioni dovreste essere in grado di individuare le singole proc e dovreste essere in grado di poter effetuare le modifiche affinchè possiate aggiungere vostre funzioni personalizzate... oppure se proprio siete cosi interessati vi risulterà più facile individuare le funzioni affinchè possano essere disassemblate per capire come funzionano.

Credo, anche qui, di aver detto tutto...

I tipi di parametri passate alle funzioni sono da interpretare secondo questa tabella (ad ogni valore va aggiunto 1, cioè 00 corrisponde al valore 01, 0C al valore 0D, ecc..).

00=INTEGER
01=TEXT
02=INTEGER
03=???
04=&REAL
05=???
06=REAL
07=&REAL
08=&INTEGER
09=&TEXT
0A=TEXT
0B=MEMORY
0C=&INTEGER
0D=000
0E=MEMORY
0F=000
10=INTEGER[]
11=???
12=000
13=000
14=000
15=000

16=???
17=&MEMORY
18=000
19=000
1A=000
1B=000
1C=000
1D=000
1E=000
1F=000
20=000
21=000
22=000
23=000

??? -> incontrato ma non so che corrispondenza abbia
000 -> mai incontrato

Eventi e alcune osservazioni

Uno script generalmente viene eseguito in modo sequenziale, questo significa che vengono interpretate tutte le sue istruzioni a partire dalla prima finendo con l'ultima e solo quando una istruzione è stata eseguita si passa ad eseguire la successiva.
Alcune volte c'è la necessità che questa normalità di esecuzione venga “denormalizzata”.
Per fare questo abbiamo la necessità di utilizzare gli eventi.
Cosa sono gli eventi?
Gli eventi sono alcune condizioni esterne e non che accadono e creano per l'appunto un evento identificabile. Ad esempio si potrebbe considerare un evento quando si preme un tasto del telecomando, il decoder riceve il segnale dal telecomando e ne interpreta i dati, guarda se è stato dichiarato di interpretare l'evento e in caso affermativo esegue il codice relativo.
Detto questo vediamo come uno script può essere denormalizzato dalla sua esecuzione sequenziale: utilizzando la funzione Background_Script (invece che la solita call_script che passa completamente il controllo allo script chiamato) si dice alla VMOS di eseguire lo script passato alla funzione solo quando lo script chiamante entra in una fase di sleep o di attesa o quando ha finito di essere eseguito, insomma, quando lo script chiamante non sta compiendo alcuna operazione (infatti si chiama background!).
Ma come fa uno script ad andare in sleep? (a parte terminare la sua esecuzione) Beh, esiste una funzione che si chiama Wait_Event e permette di aspettare che un certo evento (o più eventi) avvenga prima di continuare l'esecuzione dello script, ecco quindi che lo script entra in fase di sleep o attesa!
Gli eventi ovviamente posso essere gestiti anche in modo asincrono (ossia senza stare ad aspettare che l'evento avvenga, ma sfruttare quelli che normalmente vengono chiamati interrupts). Per gestirli infatti esistono molte funzioni, alcune sono la Put_Event, la Declare_Event, la Undeclare_Event, la Clear_To, ecc... Vi rimando a leggere la dichiarazione delle stesse per comprendere meglio come funzionano.

Funzioni Base:

Una piccola premessa solo per dire che se non sapete il nome mnemonico di una funzione potete utilizzare l'opzione di scrivere func_0xxx dove xxx è il numero esadecimale della funzione.

Per esempio se volete utilizzare la funzione 0x12C ma non ricordate che si chiama Unload_Module, beh vi basta scrivere func_012C(parametro); e il compilatore interno a Defiant ve la compilerà correttamente.

proc_0001 | Method_End

OpCode=0x0001

Come il Method_Start anche questa funzione non è una reale procedura ma solo una sorta di label/etichetta che indica la fine del codice riguardante l'evento che deve essere eseguito.

E' così dichiarata:

Method_End(INTEGER Value);

- Value: è il valore relativo al codice di fine evento (non conosco bene la logica di che valore vada realmente inserito, spesso nei firmware il valore è pari al codice presente in method_start sommato al numero di righe di codice da eseguire, ossia tutte quelle comprese tra method_start e method_end).

Vedi anche: Method_Start

proc_0003 | Method_Start

OpCode=0x0003

Non è una reale funzione, ma più che altro una sorta di label che indica dove inizia l'intercettazione di un evento nel panelscript relativo al widget per cui è stato dichiarato quel codice di evento.

E' cosi dichiarata:

Method_Start(INTEGER Value);

- Value: è il codice dell'evento a cui fa riferimento.

Vedi anche: Method_End

proc_0010 | CallScript

OpCode=0x0010

Richiama l'esecuzione di uno script, eventualmente passando anche i parametri necessari.

E' così dichiarata:

CallScript(TEXT Name, Param1, Param2, ...);

- Name: è il nome dello script da richiamare, nel caso si trovi in altro modulo si richiama in questo modo "modulo:nome_script". Il nome non contiene l'estensione del file.
- Param1, Param2, ...: in base a quanti parametri lo script si aspetta che siano passati si potranno "accodare" al nome dello script. Ovviamente il tipo di parametri passati dovrà coincidere come il numero degli stessi.

Vedi anche: Background_Script

proc_0069 | pow

OpCode=0x0069

Restituisce come risultato l'elevamento alla potenza di un numero in base all'esponente scelto.

E' così dichiarata:

REAL pow(REAL Base, REAL Exp);

- Base: è il numero che viene preso come base. Il numero che viene elevato alla potenza dell'esponente
- Exp: è l'esponente a cui elevare la base.

proc_006A | Declare_Event

OpCode=0x006A

Questa funzione associa l'esecuzione di uno script al verificarsi di un evento. Attenzione: chiamando questa procedura si cancellerà ogni evento precedentemente associato allo script.
E' dichiarata così:

Declare_Event (TEXT Script_Path, INTEGER Event_Category, INTEGER Event_Code);

- Script_Path: nome dello script senza l'estensione, nel caso si trovi in moduli differenti si aggiunge il modulo davanti separato da ":". Esempio: "basic:keya".
- Event_Category: la categoria di eventi a cui associare il codice di evento.
- Event_Code: il codice di evento che ci interessa intercettare per associarlo allo script.

Vedi anche: Undeclare_Event, Event_ID

proc_006B | Undeclare_Event

OpCode=0x006B

Dissocia l'esecuzione di uno script al verificarsi di un evento.

Undeclare_Event (INTEGER Event_Category, INTEGER Event_Code);

- Event_Category: la categoria di eventi a cui associare il codice di evento.
- Event_Code: il codice di evento che ci interessa intercettare per associarlo allo script.

Vedi anche: Declare_Event, Event_ID

proc_006D | shrt

OpCode=0x006D

Converte un valore reale in intero di tipo short (2 bytes).

E' così dichairata:

INTEGER shrt(REAL Value);

- Value: è il valore da convertire

Vedi anche: real, float, int

proc_006E | int

OpCode=0x006E

Trasforma un valore reale in intero (4 bytes).

E' dichiarata così:

INTEGER int(REAL Value);

- Value: il valore reale da convertire

Vedi anche: real, shrt, float

proc_006F | float

OpCode=0x006F

Converte un valore intero in reale di tipo float (4 bytes).

E' così dichiarata:

REAL float(INTEGER Value);

- Value: è il valore da convertire

Vedi anche: real, int, shrt

proc_0070 | real

OpCode=0x0070

Converte un valore intero restituendo il corrispondente valore reale (8 bytes).
E' dichiarata cosi:

REAL real(INTEGER Value);

- Value è il valore intero da convertire in reale

Vedi anche: float, int, shrt

proc_0071 | sin

OpCode=0x0071

Calcola il valore della funzione matematica seno di un numero.

E' così dichiarata:

REAL sin(REAL Value);

- Value: è il valore per cui trovare il seno.

Vedi anche: cos, tan, arcsin, arccos, arctan

proc_0072 | cos

OpCode=0x0072

Calcola il valore della funzione matematica coseno di un numero.

E' così dichiarata:

REAL cos(REAL Value);

- Value: è il valore per cui trovare il coseno.

Vedi anche: sin, tan, arcsin, arccos, arctan

proc_0073 | tan

OpCode=0x0073

Calcola il valore della funzione matematica tangente di un numero.

E' così dichiarata:

REAL tan(REAL Value);

- Value: è il valore per cui trovare la tangente.

Vedi anche: sin, cos, arcsin, arccos, arctan

proc_0074 | arcsin

OpCode=0x0074

Calcola il valore della funzione matematica arcoseno di un numero.

E' così dichiarata:

REAL arcsin(REAL Value);

- Value: è il valore per cui trovare l'arcoseno.

Vedi anche: sin, cos, tan, arccos, arctan

proc_0075 | arccos

OpCode=0x0075

Calcola il valore della funzione matematica arcocoseno di un numero.

E' così dichiarata:

REAL arccos(REAL Value);

- Value: è il valore per cui trovare l'arcocoseno.

Vedi anche: sin, cos, tan, arcsin, arctan

proc_0076 | arctan

OpCode=0x0076

Calcola il valore della funzione matematica arcotangente di un numero.

E' così dichiarata:

REAL arctan(REAL Value);

- Value: è il valore per cui trovare l'arcotangente.

Vedi anche: sin, cos, tan, arcsin, arccos

proc_0077 | min

OpCode=0x0077

Restituisce il valore minimo presente in una lista di valori.

E' così dichiarata:

INTEGER/REAL min(INTEGER/REAL Value1, INTEGER/REAL Value2, ...);

- Value1, Value2: è la lista di valori sulla quale calcolare il minimo

Vedi anche: max

proc_0078 | max

OpCode=0x0078

Restituisce il valore massimo presente in una lista di valori.

E' così dichiarata:

INTEGER/REAL max(INTEGER/REAL Value1, INTEGER/REAL Value2, ...);

- Value1, Value2: è la lista di valori sulla quale calcolare il massimo

Vedi anche: min

proc_0079 | ent

OpCode=0x0079

Restituisce il valore della parte intera di un numero.

E' così dichiarata:

INTEGER ent(REAL Value);

- Value: è il valore sul quale ricavare la parte intera

proc_007A | frac

OpCode=0x007A

Restituisce il valore della parte frazionaria di un numero.

E' così dichiarata:

REAL frac(REAL Value);

- Value: è il valore sul quale calcolare la parte frazionaria

proc_007B | exp

OpCode=0x007B

Restituisce il valore esponenziale di un numero.

E' così dichiarata:

REAL exp(REAL Value);

- Value: è il valore sul quale calcolare l'esponenziale

proc_007C | sqr

OpCode=0x007C

Restituisce il valore della radice quadrata di un numero.

E' così dichiarata:

REAL sqr(REAL Value);

- Value: è il valore sul quale calcolare la radice quadrata

proc_007D | ln

OpCode=0x007D

Restituisce il valore del logaritmo naturale di un numero.

E' così dichiarata:

REAL ln(REAL Value);

- Value: è il valore sul quale calcolare il logaritmo naturale

Vedi anche: log

proc_007E | log

OpCode=0x007E

Restituisce il valore del logaritmo in base 10 di un numero.

E' così dichiarata:

REAL log(REAL Value);

- Value: è il valore sul quale calcolare il logaritmo in base 10

Vedi anche: ln

proc_008D | left

OpCode=0x008D

Restituisce una sotto-stringa con il numero di caratteri specificato partendo con il conteggio dal primo carattere a sinistra.

E' così dichiarata:

TEXT left(TEXT String, INTEGER Count);

- String: è la stringa da cui estrarre la sotto-stringa
- Count: è il numero di caratteri da estrarre a partire da sinistra

Vedi anche: right, mid

proc_008E | right

OpCode=0x008E

Restituisce una sotto-stringa con il numero di caratteri specificato partendo con il conteggio dal primo carattere a destra e continuando a contare verso sinistra.

E' così dichiarata:

TEXT right(TEXT String, INTEGER Count);

- String: è la stringa da cui estrarre la sotto-stringa
- Count: è il numero di caratteri da estrarre a partire da destra

Vedi anche: left, mid

proc_008F | mid

OpCode=0x008F

Restituisce una sotto-stringa con il numero di caratteri specificato partendo con il conteggio dal carattere indice indicato.

E' così dichiarata:

TEXT mid(TEXT String, INTEGER Start, INTEGER Count);

- String: è la stringa da cui estrarre la sotto-stringa
- Start: indice del carattere da cui partire a estrarre la sotto-stringa
- Count: è il numero di caratteri da estrarre a partire dall'indice Start

Vedi anche: left, right

proc_0090 | len

OpCode=0x0090

Restituisce la lunghezza in caratteri di una stringa.

E' così dichiarata:

INTEGER len(TEXT String);

- String: è la stringa su cui calcolare la lunghezza

proc_0091 | val

OpCode=0x0091

Restituisce il valore intero corrispondente al valore scritto in stringa.

E' così dichiarata:

INTEGER val(ALPHANU String);

- String: è la stringa da convertire

Vedi anche: itos

proc_0092 | asc

OpCode=0x0092

Restituisce il codice ascii di un carattere.

E' così dichiarata:

INTEGER asc(ALPHANU Char);

- Char: è il carattere da cui ricavare il codice ascii

Vedi anche: IntToChar

proc_0093 | itos

OpCode=0x0093

Restituisce un valore intero convertito in stringa.

E' così dichiarata:

ALPHANU itos(INTEGER Value);

- Value: è il valore da convertire in stringa

Vedi anche: val

proc_0094 | IntToChar

OpCode=0x0094

Restituisce un carattere corrispondente al codice ascii passato come parametro.

E' così dichiarata:

ALPHANU IntToChar(INTEGER Value);

- Value: è il codice in ascii del carattere

Vedi anche: asc

proc_0095 | mread

OpCode=0x0095

Sconosciuta, forse legge dalla memoria

Vedi anche: mwrite

proc_0096 | mwrite

OpCode=0x0096

Sconosciuta, forse scrive in memoria

Vedi anche: mread

proc_0097 | round

OpCode=0x0097

Restituisce il valore di un numero arrotondato a N cifre dopo la virgola

E' così dichiarata:

REAL round(REAL Value, INTEGER NumberDigits);

- Value: Valore da arrotondare
- NumberDigits: Il numero di cifre da mantenere dopo la virgola del numero reale

proc_0098 | strigo

OpCode=0x0098

Imposta il modo di rappresentazione degli angoli (usato dalle funzioni per calcolare la tangente, il seno, il coseno, ecc...)

E' così dichiarata:

strigo(INTEGER Mode);

- Mode: Imposta il tipo di parametro (byte, stringa o l'unità di misura)

Vedi anche: gtrigo

proc_0099 | gtrigo

OpCode=0x0099

Restituisce il valore del modo di rappresentazione degli angoli (usato dalle funzioni per calcolare la tangente, il seno, il coseno, ecc...)

E' così dichiarata:

INTEGER gtrigo();

Vedi anche: strigo

proc_009A | peek

OpCode=0x009A

Legge un byte da un indirizzo di memoria.
Spesso è utilizzata per conoscere il valore di un byte all'interno di un array, esempio Var[i] si scrive in peek(var,i).

E' così dichiarata:

INTEGER peek(MEMORY Address, INTEGER Offset);

- Address: è l'indirizzo a cui fare riferimento
- Offset: è l'offset rispetto all'indirizzo di riferimento

Vedi anche: poke, peeks, pokes, peeki, pokei, peekb, pokeb

proc_009B | peeks

OpCode=0x009B

Legge un intero di tipo short (2 bytes) da un indirizzo di memoria.

E' così dichiarata:

INTEGER peeks(MEMORY Address, INTEGER Offset);

- Address: è l'indirizzo a cui fare riferimento
- Offset: è l'offset rispetto all'indirizzo di riferimento

Vedi anche: peek, poke, pokes, peeki, pokei, peekb, pokeb

proc_009C | peeki

OpCode=0x009C

Legge un intero (4 bytes) da un indirizzo di memoria.

E' così dichiarata:

INTEGER peeki(MEMORY Address, INTEGER Offset);

- Address: è l'indirizzo a cui fare riferimento
- Offset: è l'offset rispetto all'indirizzo di riferimento

Vedi anche: peek, poke, peeks, pokes, pokei, peekb, pokeb

proc_009D | poke

OpCode=0x009D

Scrive un byte in un indirizzo di memoria.
Spesso è utilizzato per scrivere un byte in un array. Esempio: poke(Var, i, 1), ma nella decompilazione attraverso ucomp troverete più facilmente Var[i]=1;

E' così dichiarata:

poke(MEMORY Address, INTEGER Offset, INTEGER Value);

- Address: è l'indirizzo a cui fare riferimento
- Offset: è l'offset rispetto all'indirizzo di riferimento
- Value: è il valore del byte da scrivere

Vedi anche: peek, peeks, pokes, peeki, pokei, peekb, pokeb

proc_009E | pokes

OpCode=0x009E

Scrive un intero di tipo short (2 bytes) in un indirizzo di memoria.

E' così dichiarata:

pokes(MEMORY Address, INTEGER Offset, INTEGER Value);

- Address: è l'indirizzo a cui fare riferimento
- Offset: è l'offset rispetto all'indirizzo di riferimento
- Value: è il valore da scrivere

Vedi anche: peek, poke, peeks, peeki, pokei, peekb, pokeb

proc_009F | pokei

OpCode=0x009F

Scrive un intero (4 bytes) in un indirizzo di memoria.

E' così dichiarata:

pokei(MEMORY Address, INTEGER Offset, INTEGER Value);

- Address: è l'indirizzo a cui fare riferimento
- Offset: è l'offset rispetto all'indirizzo di riferimento
- Value: è il valore da scrivere

Vedi anche: peek, poke, peeks, pokes, peeki, peekb, pokeb

proc_00A0 | peekb

OpCode=0x00A0

Legge un numero definito di bits da un indirizzo di memoria.

E' così dichiarata:

INTEGER peekb(MEMORY Address, INTEGER Offset, INTEGER Number_Bits);

- Address: è l'indirizzo a cui fare riferimento
- Offset: è l'offset rispetto all'indirizzo di riferimento, espresso in bits
- Number_Bits: è il numero di bits da leggere (un numero minore o uguale a 32)

Vedi anche: peek, poke, peeks, pokes, peeki, pokei, pokeb

proc_00A1 | pokeb

OpCode=0x00A1

Scrive un un numero definito di bits in un indirizzo di memoria.

E' così dichiarata:

pokeb(MEMORY Address, INTEGER Offset, INTEGER Number_Bits, INTEGER Value);

- Address: è l'indirizzo a cui fare riferimento
- Offset: è l'offset rispetto all'indirizzo di riferimento, espresso in bits
- Number_Bits: è il numero di bits da scrivere (un numero minore o uguale a 32 bit)
- Value: è il valore da scrivere

Vedi anche: peek, poke, peeks, pokes, peeki, pokei, peekb

proc_00A2 | smode

OpCode=0x00A2

Imposta un parametro di esecuzione nella virtual machine dedicata al pantalk (Non ho ancora capito che scopo ha questa funzione ma ecco qui cosa in merito, sembra che venga utilizzata per impostare alcuni parametri di funzioni come peek, poke ecc...)

E' così dichiarata:

smode(INTEGER Att, INTEGER Domain, INTEGER Value);

- Att: Imposta il tipo di parametro (byte, stringa o l'unità di misura)
- Domain: Imposta il dominio di che cosa va a modificare (solo script corrente, tutti gli script, solo il modulo corrente)
- Value: Valore del parametro

Vedi anche: gmode

proc_00A3 | gmode

OpCode=0x00A3

Restituisce il valore di un parametro di esecuzione nella virtual machine dedicata al pantalk (Guardate smode per capire un pò meglio)

E' così dichiarata:

INTEGER gmode(INTEGER Att, INTEGER Domain);

- Att: Imposta il tipo di parametro (byte, stringa o l'unità di misura)
- Domain: Imposta il dominio di che cosa va a modificare (solo script corrente, tutti gli script, solo il modulo corrente)

Vedi anche: smode

proc_00A4 | HexToInt

OpCode=0x00A4

Restituisce una stringa esadecimale convertito in intero.

E' così dichiarata:

INTEGER HexToInt(ALPHANU Value);

- Value: è la stringa da convertire

Vedi anche: IntToHex

proc_00A5 | IntToHex

OpCode=0x00A5

Restituisce un numero intero convertito in stringa esadecimale.

E' così dichiarata:

ALPHANU IntToHex(INTEGER Value);

- Value: è il valore da convertire

Vedi anche: HexToInt

proc_00A6 | SizeOf

OpCode=0x00A6

Restituisce la dimensione di una variabile in bytes.

E' così dichiarata:

INTEGER SizeOf(ANYTYPE Var);

- Var: Imposta la variabile sulla quale calcolare la dimensione

proc_00A7 | VCSparam

OpCode=0x00A7

Sconosciuta

proc_00A8 | OpenStream

OpCode=0x00A8

Sconosciuta, dovrebbe servire per aprire uno stream, come ad esempio un file

Funzioni Supplementari:

proc_012C | Unload_Module

OpCode=0x012C

Chiude un modulo liberandolo dalla memoria senza richiamare lo script di finalizzazione.

E' così dichiarata:

Unload_Module(INTEGER Module_Id, &INTEGER Error_Code);

- Module_Id: è l'ID del modulo da chiudere
- Error_Code: restituisce il codice di errore nel caso fosse presente

Vedi anche: Load_Module, Module_End

proc_012D | Copy_Adr_To_Int

OpCode=0x012D

Converte un indirizzo di memoria in un numero intero.

E' così dichiarata:

Copy_Adr_To_Int(MEMORY Mem_Adr, &INTEGER Adr_Value);

- Mem_Adr: è l'indirizzo da convertire in intero
- Adr_Value: restituisce il valore dell'indirizzo in formato INTEGER

Vedi anche: Copy_Int_To_Adr

proc_012E | Videotext_putc

OpCode=0x012E

Sconosciuta

E' così dichiarata:

Videotext_putc(??? param);

proc_012F | DfNibbleOper

OpCode=0x012F

Sconosciuta

E' così dichiarata:

DfNibbleOper(MEMORY param1, INTEGER param2, INTEGER param3, INTEGER param4, MEMORY param5, INTEGER param6, &INTEGER param7);

proc_0130 | Background_Script

OpCode=0x0130

Chiama uno script da eseguire in modalità “background”.

E' così dichiarata:

Background_Script(TEXT Script_Name);

- Script_Name: è lo script da eseguire in background

Vedi anche: Call_Script

proc_0131 | Flash_Item_Write

OpCode=0x0131

Scrive un Item in un record nella flash.

E' così dichiarata:

Flash_Item_Write(TEXT Record_Name, TEXT Item_Name, MEMORY Buf_Adr, INTEGER Buf_Size, &INTEGER Error_Code);

- Record_Name: è il nome del record nel quale scrivere l'Item
- Item_Name: è il nome dell'Item da scrivere nel record
- Buf_Adr: è il buffer che contiene i dati dell'Item
- Buf_Size: è la dimensione del buffer
- Error_Code: restituisce il codice di errore

Vedi anche: Flash_Item_Read, Flash_Item_Info

proc_0132 | Flash_Item_Info

OpCode=0x0132

Acquisisce informazioni su un Item in un record in flash.

E' così dichiarata:

Flash_Item_Info(TEXT Record_Name, TEXT Item_Name, &MEMORY Item_Adr, &INTEGER Item_Size, &INTEGER Error_Code);

- Record_Name: è il nome del record nel quale cercare l'Item
- Item_Name: è il nome dell'Item da cercare nel record
- Item_Adr: è l'indirizzo dei dati dell'Item in flash
- Item_Size: è la dimensione dei dati dell'Item
- Error_Code: restituisce il codice di errore

Vedi anche: Flash_Item_Read, Flash_Item_Write

proc_0133 | Flash_Module_Write

OpCode=0x0133

Scrive un Modulo in un record in flash.

E' così dichiarata:

Flash_Module_Write(TEXT Module_Name, &INTEGER Error_Code);

- Module_Name: è il nome del Modulo da scrivere in flash
- Error_Code: restituisce il codice di errore

proc_0134 | Flash_Text_Write

OpCode=0x0134

Scrive un testo in un Item in flash.

E' così dichiarata:

Flash_Text_Write(TEXT Record_Name, TEXT Item_Name, TEXT Text, INTEGER Count, &INTEGER Error_Code);

- Record_Name: è il nome del record nel quale scrivere il testo
- Item_Name: è il nome dell'Item testo in cui scrivere
- Text: è il testo da scrivere
- Count: è la lunghezza del testo da scrivere
- Error_Code: restituisce il codice di errore

Vedi anche: Flash_Text_Read

proc_0135 | Download_Device_Volume

OpCode=0x0135

Scarica la volume module list.

E' così dichiarata:

Download_Device_Volume(TEXT Device_Volume_Name, &INTEGER Download_id, &INTEGER Error_Code);

- Device_Volume_Name: è il nome del Volume da scaricare
- Download_id: è l'id dell'evento generato
- Error_Code: restituisce il codice di errore

Device_Volume_Name = "MLOAD", "LCARD", "RCARD", "MODEM", "SERIAL", "PARALLEL".
Questa funzione crea un evento quando finisce di scaricare la volume module list, l'evento da intercettare ha come codice il valore restituito da Download_id.

Vedi anche: Download_Volume_Module

proc_0136 | Download_Volume_Module

OpCode=0x0136

Scarica il volume module.

E' così dichiarata:

Download_Volume_Module(INTEGER Volume_id, TEXT Module_Name, &INTEGER Download_id, &INTEGER Error_Code);

- Volume_id: è l'id del Volume da scaricare
- Module_Name: è il nome del Module da scaricare
- Download_id: è l'id dell'evento generato
- Error_Code: restituisce il codice di errore

Questa funzione crea un evento quando finisce di scaricare il volume module, l'evento da intercettare ha come codice il valore restituito da Download_id.

Vedi anche: Download_Device_Volume, Abort_Module_Download

proc_0137 | Panel_func

OpCode=0x0137

Sconosciuta

E' così dichiarata:

Panel_func(INTEGER param);

proc_0138 | NullSubDSX7071_138

OpCode=0x0138

Sconosciuta

E' così dichiarata:

NullSubDSX7071_138(&MEMORY param1, TEXT param2);

proc_0139 | NullSubDSX7071_139

OpCode=0x0139

Sconosciuta

E' così dichiarata:

NullSubDSX7071_139(&INTEGER param1, ??? param2, &INTEGER param3, ??? param4, &REAL param5, &TEXT param6, MEMORY param7);

proc_013A | NullSubDSX7071_13A

OpCode=0x013A

Sconosciuta

E' così dichiarata:

NullSubDSX7071_13A(INTEGER[] param1, ??? param2, ??? param3);

proc_013B | Get_Po_Slider

OpCode=0x013B

Restituisce il valore del widget slider.

E' così dichiarata:

Get_Po_Slider(INTEGER Panel_id, TEXT Widget_Name, INTEGER Action_Flag, &REAL Min, REAL Max, TEXT);

Mnemonics:

La costante $CURRENT ha valore pari a -1, viene utilizzata per indicare in automatico l'id del pannello corrente o dati simili all'id del pannello.

Attribute ID

$BKG_COLOR_ACTIVE = 0
$BORDER_COLOR_ACTIVE = 1
$STATE = 3
$TEXT_COLOR_ACTIVE = 4
$JUSTIFICATION = 6
$GROUP = 8
$KEY_EVENT = 11
$BORDER_STYLE_ACTIVE = 18
$BKG_COLOR_ST_0 = 52
$BKG_COLOR_ST_1 = 53
$BORDER_STYLE_ST_0 = 54
$BORDER_STYLE_ST_1 = 55
$NOT_DRAW = 56
$INVISIBILITY = 58
$BORDER_COLOR_ST_0 = 60
$BORDER_COLOR_ST_1 = 61
$TEXT_COLOR_ST_0 = 62
$TEXT_COLOR_ST_1 = 63
$BKG_COLOR_ST_2 = 64
$BORDER_STYLE_ST_2 = 65
$BORDER_COLOR_ST_2 = 66
$TEXT_COLOR_ST_2 = 67
$BORDER_WIDTH = 83
$BORDER_WIDTH_A = 84
$BORDER_WIDTH_B = 85
$BORDER_INTER = 86
$BORDER_INTER_A = 87
$BORDER_INTER_B = 88
$PAGE = 90
$PRIORITY = 92

Char Constant Code

$Symb_A = 10
$Symb_D = 13
$Symb_5C = 92
$Symb_7F = 127
$Symb_80 = 128
$Symb_Subtitle = 141
$Symb_Favorite = 142
$Symb_Locked = 143
$Symb_UnLock = 144
$Symb_Point = 149
$Symb_ArrowRight = 154
$Symb_A0 = 160
$Symb_ArrowUp = 170
$Symb_ArrowLeft = 172

$Symb_ArrowDown = 186

Command ID

$LCARD_HISTO = 0
$LCARD_STATE = 1
$LCARD_INPUT = 2
$LCARD_OUTPUT = 3
$LCARD_RESET = 4
$LCARD_INOUT = 5
$LCARD_SETUP = 6
$LCARD_SEND = 7
$LCARD_ABORT = 8
$LCARD_PPS_REQ = 9
$RCARD_HISTO = 16
$RCARD_STATE = 17
$RCARD_INPUT = 18
$RCARD_OUTPUT = 19
$RCARD_RESET = 20
$RCARD_INOUT = 21
$RCARD_SETUP = 22
$RCARD_SEND = 23
$RCARD_ABORT = 24
$RCARD_PPS_REQ = 25
$Emm_opi_table = 32
$Emm_get_table = 33
$Emm_start = 34
$Emm_stop = 35
$NOTIFY_SEND = 36
$NOTIFY_GET = 37
$SCTV_SND_LVL_INFO = 48
$SCTV_SET_LVL_SND = 49
$SCTV_WRT_LVL_SND = 50
$SCTV_MUTE_SET = 51
$SCTV_SND_TYPE_INFO = 52
$SCTV_SET_SND_TYPE = 53
$SCTV_WRT_SND_TYPE = 54
$SCTV_TV_INFO = 55
$SCTV_SET_TV = 56
$SCTV_WRT_TV = 57
$SCTV_SET_HD_FORMAT = 58
$SCTV_WRT_HD_FORMAT = 59
$SCTV_HD_FORMAT_INFO = 60
$SCTV_HD_INFO = 61
$SCTV_DSWITCH_INFO = 62
$SCTV_RGB_INFO = 64
$SCTV_SET_RGB = 65
$SCTV_WRT_RGB = 66
$SCTV_SWITCHING_INFO = 67

$SCTV_SET_SWITCHING = 68
$SCTV_BYPSS_INFO = 69
$SCTV_SET_BYPSS = 70
$SCTV_SIG_RGB_INFO = 71
$SCTV_SIG_RGB_SET = 72
$SCTV_WRT_SWITCHING = 73
$SCTV_SIG_RGB_WRT = 74
$SCTV_SET_16_9 = 75
$SCTV_INFO = 76
$SCTV_AC_OUTLET_SET = 77
$SCTV_AC_OUTLET_WRT = 78
$SCTV_AC_OUTLET_INFO = 79
$SCVCR_VIDEO_INFO = 80
$SCVCR_SET_VIDEO = 81
$SCVCR_WRT_VIDEO = 82
$SCVCR_COMMUT_INFO = 83
$SCVCR_SND_TYPE_INFO = 84
$SCVCR_SET_SND_TYPE = 85
$SCVCR_WRT_SND_TYPE = 86
$SCVCR_BYPSS_INFO = 87
$SCVCR_SET_BYPSS = 88
$SCVCR_INFO = 89
$TUNER_FREQ_INFO = 96
$TUNER_AGC_READ = 97
$TUNER_BER_READ = 98
$TUNER_START_SCANNING = 99
$TUNER_NEXT_SCANNING = 100
$TUNER_TUNING_INFO = 101
$TUNER_TUNING_SET = 102
$TUNER_STOP_SCANNING = 103
$TUNER_SET_DUAL_SWITCH = 104
$TUNER_WRT_DUAL_SWITCH = 105
$TUNER_GET_DUAL_SWITCH = 106
$TUNER_DEMOD_INFO = 107
$TUNER_SET_SYMBOL = 108
$TUNER_GET_SYMBOL = 109
$TUNER_SET_BTSC_MODE = 110
$TUNER_GET_BTSC_MODE = 111
$MODEM_DTMF_DIAL = 112
$MODEM_SEND = 114
$MODEM_DECONNEXION = 115
$MODEM_SEND_ENQ = 116
$MODEM_SET = 117
$MODEM_SET_LINK = 118
$MODEM_GET_LINK = 119
$MODEM_STOP = 120
$SERIAL_SETUP = 129
$SERIAL_SEND = 130

$SERIAL_HWR_CTRL = 131
$SERIAL_SET_CTRL = 132
$SERIAL_GET_LINK = 133
$SERIAL_SET_LINK = 134
$SERIAL_STOP = 135
$PARALLEL_SEND = 145
$PARALLEL_SETUP = 146
$PARALLEL_SET_CTRL = 147
$PARALLEL_GET_LINK = 148
$PARALLEL_SET_LINK = 149
$PARALLEL_STOP = 150
$TUNER_INIT_VARIABLE = 154
$TUNER_GET_VARIABLE = 155
$TUNER_MULTI_TS_INFO = 156
$TUNER_SET_TS = 157
$TUNER_GET_TS = 158
$TUNER_SET_SOURCE = 159
$SOUND_AUDIO_BIP = 160
$SOUND_AUDIO_SAMPLE_PLAY = 161
$SOUND_AUDIO_SAMPLE_STOP = 163
$AUDIO_PLAY = 164
$AUDIO_ADD_DATA = 165
$AUDIO_PAUSE = 166
$AUDIO_RESTART = 167
$AUDIO_FFW = 168
$AUDIO_STOP = 169
$AUDIO_INFO = 170
$SOUND_AUDIO_INFO_CHANNEL = 171
$SOUND_AUDIO_SELECT_CHANNEL = 172
$SOUND_BUZZER_AUDIO_PROLOGIC_SET = 173
$SOUND_BUZZER_INFO_AUDIO_PROLOGIC_WRT = 174
$AUDIO_PROLOGIC_INFO = 175
$POWER_ON_INFO = 176
$POWER_STDBY_ON = 177
$POWER_ON_WRT_CONFIG = 178
$POWER_ON_CONFIG_INFO = 179
$POWER_RESET = 180
$POWER_ECO_RUNNING = 181
$POWER_ECO_INFO = 182
$CLOCK_TIME_INFO = 192
$CLOCK_TIME_SET = 193
$CLOCK_OFFSET_GMT = 194
$CLOCK_ALARM_SET = 195
$CLOCK_WAKE_UP_SET = 196
$CLOCK_ALARM_LIST = 197
$CLOCK_ALARM_INFO = 198
$CLOCK_ALARM_MODIF = 199
$CLOCK_ALARM_DELETE = 200

$CLOCK_USER_DELETE = 201
$CLOCK_TIMER_SET = 202
$CLOCK_TIMER_STOP = 203
$CLOCK_TIMER_REARM = 204
$CLOCK_TIMER_REMOVE = 205
$CLOCK_TIMER_INFO = 206
$DISPLAY_SET_BITMAP = 208
$DISPLAY_SET_HOUR = 209
$DISPLAY_SET_DISPLAY = 210
$DISPLAY_SET_LED = 211
$DISPLAY_GET_LED = 212
$DISPLAY_INFO = 213
$DISPLAY_CLEAR = 214
$DISPLAY_CLEAR_LINE = 215
$DISPLAY_TUNING = 216
$DISPLAY_GET_DM = 217
$DISPLAY_SET_DM = 218
$DISPLAY_GET_SYMBOL = 219
$DISPLAY_SET_SYMBOL = 220
$DISPLAY_GET_GRAPHIC = 221
$DISPLAY_SET_GRAPHIC = 222
$DISPLAY_STRING = 223
$BACKUP_INFO_STRUCT = 224
$BACKUP_SECTOR_ADR_WRITE = 225
$BACKUP_SECTOR_WRITE = 226
$BACKUP_SECTOR_READ = 227
$BACKUP_SECTOR_ERASE = 228
$DISPLAY_DEF_CHAR = 229
$AUDIO_SET_BUZZER = 230
$AUDIO_BUZZER_INFO = 231
$AUDIO_GET_MODE = 232
$AUDIO_SET_MODE = 233
$AUDIO_SELECT_SAP = 234
$AUDIO_INFO_SAP = 235
$AUDIO_INPUT = 236
$AUDIO_SET_DIGITAL_OUTPUT = 237
$AUDIO_WRT_DIGITAL_OUTPUT = 238
$AUDIO_DIGITAL_OUTPUT_INFO = 239
$MASS_STORAGE_STRUCT = 240
$MASS_STORAGE_SECTOR_ADR = 241
$MASS_STORAGE_SECTOR_WRITE = 242
$MASS_STORAGE_SECTOR_READ = 243
$MASS_STORAGE_SECTOR_ERASE = 244
$MASS_STORAGE_WRITE = 245
$MASS_STORAGE_READ = 246
$MASS_STORAGE_ADD_DATA = 247
$MASS_STORAGE_STOP = 248
$MASS_STORAGE_ADD_FRAGMENT = 249

$MASS_STORAGE_REQ_INFO = 250
$DISPLAY_SET_LUMINOSITY = 251
$DISPLAY_GET_LUMINOSITY = 252
$MLOAD_SECTION = 256
$MLOAD_SECTION_ALL = 257
$MLOAD_TABLE_LOAD = 258
$MLOAD_GROUP_LOAD = 259
$MLOAD_INFO = 260
$MLOAD_GROUP_ADD = 261
$MLOAD_GROUP_SUP = 262
$MLOAD_DESCRIPT = 263
$MLOAD_DELETE = 264
$MLOAD_DELETE_ALL = 265
$MLOAD_LOAD_ABORT = 266
$MLOAD_DIRECTORY = 267
$MLOAD_MODULE_LOAD = 268
$MLOAD_SEGMENT_LOAD = 269
$MLOAD_INFO_SEGMENT = 270
$MLOAD_DESCRIPT_SEGMENT = 271
$SERVICE_PID_INFO = 272
$SERVICE_SET_PID = 273
$SERVICE_DEMUX_PID = 274
$SERVICE_DEMUX_INFO = 275
$SERVICE_ECM_INFO = 276
$SERVICE_SET_ECM = 277
$SERVICE_ADD_ECM = 278
$SERVICE_DEL_ECM = 279
$SERVICE_STOP_PID = 280
$SERVICE_START = 281
$SERVICE_STOP = 282
$SERVICE_MODIFY = 283
$SERVICE_FREEZE = 285
$SERVICE_PLAY = 286
$SERVICE_BLK_SCREEN = 287
$SERVICE_FORMAT_INFO = 288
$SEVIRCE_DEL_ALL_PICTURE = 289
$SEVIRCE_DEL_PICTURE = 290
$SEVIRCE_DISPLAY_PICTURE = 291
$SERVICE_SET_SUBTITLES = 292
$SERVICE_STOP_SUBTITLES = 293
$SERVICE_FREE_PICTURE = 294
$SERVICE_INIT_VARIABLE = 295
$SERVICE_GET_VARIABLE = 296
$SERVICE_SET_VIDEO_PARAM = 298
$SERVICE_GET_VIDEO_PARAM = 299
$SERVICE_STATUS = 300
$SCAUX_COMMUT_INFO = 325
$SCAUX_BYPSS_INFO = 326

$SCAUX_SET_BYPSS = 327
$SCAUX_VIDEO_INFO = 328
$SCAUX_SET_VIDEO = 329
$SCAUX_WRT_VIDEO = 330
$SCAUX_SND_TYPE_INFO = 331
$SCAUX_SET_SND_TYPE = 332
$SCAUX_WRT_SND_TYPE = 333
$SERVICE_DISPLAY_IMAGE = 336
$REMOD_SET_CHNL = 352
$REMOD_WRT_CHNL = 353
$REMOD_INFO_CHNL = 354
$REMOD_SET_SIGNAL = 355
$REMOD_WRT_SIGNAL = 356
$REMOD_INFO_SIGNAL = 357
$REMOD_SET_HF = 358
$REMOD_WRT_HF = 359
$REMOD_INFO_HF = 360
$MCOM_FREE_DEMUX_CHAN = 366
$MCOM_NEW_DEMUX_CHAN = 367
$CA_INFO = 368
$CA_SELECT = 369
$CA_SET_INFO = 370
$CA_GET_INFO = 371
$CA_SET_CAT = 372
$CA_EMM_START = 373
$CA_EMM_STOP = 374
$CA_ECM_INFO = 375
$CA_MODIFY_ECM = 376
$CA_ADD_ECM = 377
$CA_DEL_ECM = 378
$CA_PID_LIST = 380
$CA_DESCR_LIST = 381
$CA_DEL_ALL_ECM = 382
$CA_ECM_CONTROL = 383
$CA_SET_EMM = 384
$CA_GET_EMM = 385
$CA_SELECT_CARD = 386
$CA_GET_CARD = 387
$UNIMODEM_SET = 400
$UNIMODEM_COMM_SET = 401
$UNIMODEM_EVENT_PATTERN = 402
$UNIMODEM_ACK_PATTERN = 403
$UNIMODEM_NACK_PATTERN = 404
$UNIMODEM_CHAR_SET = 405
$UNIMODEM_SET_CTRL = 406
$UNIMODEM_CMD = 407
$UNIMODEM_SEND = 408
$UNIMODEM_ACTION = 409

$UNIMODEM_DISCONNECT = 410
$UNIMODEM_SET_LINK = 411
$UNIMODEM_GET_LINK = 412
$UNIMODEM_STOP_MCOM_FREE_PORT = 413
$MCOM_GET_FILTER = 414
$MCOM_PACKET_RESET = 415
$MCOM_INIT_DEMUX = 416
$MCOM_INIT_COMM = 417
$MCOM_INIT_FILTER = 418
$MCOM_CHANGE_FILTER = 419
$MCOM_PACKET_DEL = 420
$MCOM_PACKET_HEADER = 421
$MCOM_PACKET_END = 423
$MCOM_PACKET_FCS = 424
$MCOM_PACKET_ESC = 425
$MCOM_START = 426
$MCOM_RESTART = 427
$MCOM_STATUS = 428
$MCOM_STOP = 429
$MCOM_SEND = 430
$MCOM_SET_CTRL = 431
$BUS_1394_SET = 432
$BUS_1394_INF0_PERIPH = 433
$BUS_1394_INFO = 434
$BUS_1394_ALLOC_CHANNEL = 435
$BUS_1394_INFO_CHANNEL = 436
$BUS_1394_FREE_CHANNEL = 437
$BUS_1394_OPEN_CONNECT = 438
$BUS_1394_CLOSE_CONNECT = 439
$BUS_1394_LIST_CONNECT = 440
$BUS_1394_INFO_CONNECT = 441
$BUS_1394_RESET = 442
$BUS_1394_SEND_FCP = 443
$BUS_1394_SPEED_PERIPH = 444
$BUS_1394_ISOCH_LISTEN = 445
$BUS_1394_ISOCH_TALK = 446
$BUS_1394_ISOCH_STOP = 447
$TS_REMUX_SET_PID = 448
$TS_REMUX_PID_INFO = 449
$TS_REMUX_ADD_PID = 450
$TS_REMUX_STOP_PID = 451
$TS_REMUX_SET_SCR_PID = 452
$TS_REMUX_ADD_SCR_PID = 453
$TS_REMUX_CHANGE_SCR_PID = 454
$TS_REMUX_SCR_PID_INFO = 455
$TS_REMUX_SET_DATA_PID = 456
$TS_REMUX_ADD_DATA_PID = 457
$TS_REMUX_STOP_DATA_PID = 458

$TS_REMUX_SET_SECTION = 459
$TS_REMUX_ADD_SECTION = 460
$TS_REMUX_STOP_SECTION = 461
$TS_REMUX_SECTION_INFO = 462
$TS_REMUX_START = 463
$TS_REMUX_STOP = 464
$BUS_1394_ASYNC_WRITE = 474
$BUS_1394_ASYNC_READ = 475
$BUS_1394_ASYNC_LOCK = 476
$KEYBOARD_INIT = 480
$KEYBOARD_SET_KEYS_TYPE = 481
$KEYBOARD_GET_KEYS_TYPE = 482
$KEYBOARD_SET_MOD_MASKS = 483
$KEYBOARD_GET_MOD_MASKS = 484
$KEYBOARD_SET_LINK_LEDS = 485
$KEYBOARD_GET_LINK_LEDS = 486
$KEYBOARD_SET_LEDS = 487
$KEYBOARD_GET_LEDS = 488
$KEYBOARD_CHANGE_LEDS = 489
$KEYBOARD_SET_MODIFIERS = 490
$KEYBOARD_GET_MODIFIERS = 491
$KEYBOARD_SET_REPEAT_CONF = 492
$KEYBOARD_GET_REPEAT_CONF = 493
$KEYBOARD_SET_AUTO_REPEAT = 494
$KEYBOARD_GET_AUTO_REPEAT = 495
$KEYBOARD_INFO = 496
$KEYBOARD_ATTACH = 497
$NET_SOCKET_OPEN = 512
$NET_SOCKET_CLOSE = 513
$NET_SOCKET_BIND = 514
$NET_SOCKET_CONNECT = 515
$NET_SOCKET_SHUTDOWN = 516
$NET_SOCKET_LISTEN = 517
$NET_SOCKET_ACCEPT = 518
$NET_SOCKET_SEND = 519
$NET_SOCKET_RECV_START = 520
$NET_SOCKET_RECV_BUFF = 521
$NET_SOCKET_RECV_STOP = 522
$NET_SOCKET_SETOPT = 523
$NET_SOCKET_GETOPT = 524
$NET_SOCKET_GETADR = 525
$NET_SOCKET_RECV_FLUSH = 526
$NET_SOCKET_SENDTO = 527
$NET_STACK_SET_IFCONFIG = 528
$NET_STACK_GET_IFCONFIG = 529
$NET_STACK_UP = 530
$NET_STACK_DOWN = 531
$NET_STACK_STATE = 532

$NET_STACK_ADDROUTE = 533
$NET_STACK_GET_STAT = 534
$NET_STACK_DELROUTE = 535
$NET_STACK_GETROUTE = 536
$NET_STACK_PPP_OPEN = 537
$NET_STACK_PPP_CLOSE = 538
$NET_STACK_PPP_SETCONFIG = 539
$NET_STACK_PPP_GETCONFIG = 540
$NET_STACK_PPP_SETSECRET = 541
$NET_STACK_PPP_GETSECRET = 542
$NET_STACK_BOOTP = 543
$NET_PORT_ATTACH = 544
$NET_PORT_DETTACH = 545
$NET_PORT_GETCONFIG = 546
$NET_DNS_SET_TIMEOUT = 553
$NET_DNS_GET_TIMEOUT = 554
$NET_DNS_SET_SERVER_ADR = 555
$NET_DNS_GET_SERVER_ADR = 556
$NET_DNS_GET_HOST_BY_ADR = 557
$NET_DNS_GET_HOST_BY_NAME = 558
$NET_STACK_PING = 559
$POINTER_GET_INFO = 560
$POINTER_DEFINE_CLUT = 561
$POINTER_GET_CLUT = 562
$POINTER_DEFINE_PIXMAP = 563
$POINTER_SET_CURSOR = 564
$POINTER_GET_CURSOR = 565
$POINTER_SHOW = 566
$POINTER_HIDE = 567
$POINTER_LINK_MOUSE = 568
$POINTER_GET_MOUSE = 569
$POINTER_SET_POSITION = 570
$POINTER_GET_POSITION = 571
$POINTER_CREATE_AREA = 572
$POINTER_DELETE_AREA = 573
$POINTER_SET_AREA_ATT = 576
$POINTER_GET_INFO_AREA = 577
$POINTER_MOVE_AREA = 578
$POINTER_RESIZE_AREA = 579
$POINTER_SET_SPEED = 580
$POINTER_GET_SPEED = 581
$POINTER_WRT_SPEED = 582
$POINTER_ADD_PIXAMP = 583
$POINTER_DEL_PIXMAP = 584
$POINTER_GET_PIXMAP = 585
$POINTER_DEFINE_LOG_BUTTONS = 586
$POINTER_GET_LOG_BUTTONS = 587
$POINTER_RESTACK_AREA = 588

$POINTER_ATTACH = 589
$POINTER_SET_RESOLUTION = 590
$POINTER_GET_RESOLUTION = 591
$DEMUX_SET_SOURCE = 592
$DEMUX_INFO_SOURCE = 593
$DEMUX_SET_PID = 594
$DEMUX_INFO_PID = 595
$DEMUX_START = 596
$DEMUX_STOP = 597
$DEMUX_INFO_DEMUX = 598
$DEMUX_LOAD_TP = 599
$DEMUX_LOAD_TP_ALL = 600
$DEMUX_LOAD_PES = 601
$DEMUX_LOAD_PES_ALL = 602
$DEMUX_LOAD_ABORT = 603
$DEMUX_GET_BUFFER = 604
$DEMUX_GET_PCR = 605
$POINTER_GET_AREA_LIST = 608
$LOADER_PF_WRITE = 624
$LOADER_PF_READ = 625
$LOADER_PF_STATUS = 626
$LOADER_REQ_UPDATE = 627
$LOADER_REQ_STATUS = 628
$LOADER_REQ_CANCEL = 629
$VIDEO_ADD_DATA = 638
$VIDEO_DISPLAY_INFO = 639
$VIDEO_CREATE_SEQUENCE = 656
$VIDEO_DEL_SEQUENCE = 657
$VIDEO_INFO_SEQUENCE = 658
$VIDEO_CHANGE_DEST = 659
$VIDEO_FREEZE = 660
$VIDEO_CONTINUE = 661
$VIDEO_STOP = 662
$VIDEO_SEQUENCE_LIST = 663
$VIDEO_CHANGE_MODE = 664
$VIDEO_CHANGE_BLENDING = 665
$VIDEO_CHANGE_RESIZING = 666
$VIDEO_CHANGE_SRC = 667
$VIDEO_FORMAT_INFO = 668
$VIDEO_FORMAT_MPEG_LAYER_INFO_HD = 669
$VIDEO_SET_DISPLAY = 670
$VIDEO_WRT_DISPLAY = 671
$SUBTITLE_START = 672
$SUBTITLE_STOP = 673
$SUBTITLE_CHANGE_BUFFER = 674
$SUBTITLE_INFO = 675
$SUBTITLE_CHANGE_PAGE = 676
$SUBTITLE_REFRESH = 677

$SUBTITLE_SETUP = 678
$SUBTITLE_ADD_DATA = 679
$SUBTITLE_PAUSE = 680
$SUBTITLE_RESUME = 681
$VIDEO_SET_RESOLUTION = 683
$VIDEO_GET_RESOLUTION = 684
$VIDEO_PAUSE = 685
$VIDEO_RESUME = 686
$VIDEO_CHANGE_SPEED = 687
$PICTURE_LAYER_INFO = 704
$PICTURE_SET_LAYER_DISPLAY = 705
$PICTURE_GET_LAYER_DISPLAY = 706
$PICTURE_DECOMPRESS = 707
$PICTURE_ADD_DATA = 708
$PICTURE_FROM_LAYER = 709
$PICTURE_DISPLAY = 710
$PICTURE_CLEAR_RECT = 711
$PICTURE_DELETE_IMAGE = 712
$PICTURE_DELETE_GROUP = 713
$PICTURE_CLEAR_ALL = 714
$PICTURE_FILL_RECT = 715
$PICTURE_COPY_AREA = 716
$PICTURE_INFO = 717
$PICTURE_GROUP_INFO = 718
$PICTURE_SUSPEND_GROUP = 719
$PICTURE_RESUME_GROUP = 720
$PICTURE_STOP_GROUP = 721
$PICTURE_GET_LAYER_ORDER = 722
$PICTURE_GET_LAYER_ORDER = 723
$PICTURE_SET_RESOLUTION = 724
$PICTURE_GET_RESOLUTION = 725
$TIMER_SETUP = 736
$TIMER_STOP = 737
$TIMER_REARM = 738
$TIMER_REMOVE = 739
$TIMER_INFO = 740
$GRAPHIC_DECOMP_ORIGINAL = 752
$GRAPHIC_DECOMP_TO_RGB = 753
$GRAPHIC_DECOMP_TO_CLUT = 754
$GRAPHIC_ADD_DATA = 755
$GRAPHIC_SUSPEND_DECOMP = 756
$GRAPHIC_RESUME_DECOMP = 757
$GRAPHIC_STOP_DECOMP = 758
$GRAPHIC_CONVERT_PIXMAP = 759
$GRAPHIC_ABORT_CONVERT = 760
$GRAPHIC_NOTIFY = 761
$GRAPHIC_CREATE_BATCH = 762
$GRAPHIC_PLAY_BATCH = 763

$GRAPHIC_CONTINUE_BATCH = 764
$GRAPHIC_SUSPEND_BATCH = 765
$GRAPHIC_RESUME_BATCH = 766
$GRAPHIC_DELETE_BATCH = 767
$DVB_SUBTITLE_SETUP = 768
$DVB_SUBTITLE_START = 769
$DVB_SUBTITLE_STOP = 770
$DVB_SUBTITLE_ADD_DATA = 771
$DVB_SUBTITLE_PAUSE = 772
$DVB_SUBTITLE_RESUME = 773
$DVB_SUBTITLE_FLUSH = 774
$DVB_SUBTITLE_SAVE_OBJECT = 775
$DVB_SUBTITLE_DISPLAY_PAGE = 776
$GRAPHIC_STOP_ALL = 781
$GRAPHIC_CAPTURE_PIXMAP = 782
$GRAPHIC_FROM_LAYER = 783
$SERVICE_GET_PCR = 784
$SERVICE_PROGRAMM_PCR_SYNCHRO = 785
$SERVICE_REMOVE_PCR_SYNCHRO = 786
$SERVICE_INFO_PCR_SYNCHRO = 787
$SERVICE_ANTI_COPY_SET = 788
$SERVICE_ANTI_COPY_WRT = 789
$SERVICE_ANTI_COPY_INFO = 790
$SERVICE_ANTI_COPY_UPDATE = 791
$SERVICE_ANTI_COPY_OFF = 792
$SERVICE_CAPTION = 793
$SERVICE_SET_SYNCHRO = 794
$SERVICE_GET_SYNCHRO = 795
$SERVICE_PAUSE = 796
$SERVICE_RESUME = 797
$DVB_CI_INIT_COM = 800
$DVB_CI_INFO_CONNECT = 801
$DVB_CI_SEND_SPDU = 802
$DVB_CI_CREATE_CONNECT = 803
$DVB_CI_DELETE_CONNECT = 804
$POD_EXT_CHANNEL_SEND = 805
$POD_EXT_CHANNEL_STOP = 806
$NET_MLOAD_SECTION = 816
$NET_MLOAD_SECTION_ALL = 817
$NET_MLOAD_TABLE_LOAD = 818
$NET_MLOAD_GROUP_LOAD = 819
$NET_MLOAD_INFO = 820
$NET_MLOAD_GROUP_ADD = 821
$NET_MLOAD_GROUP_SUP = 822
$NET_MLOAD_DESCRIPT = 823
$NET_MLOAD_DELETE = 824
$NET_MLOAD_DELETE_ALL = 825
$NET_MLOAD_LOAD_ABORT = 826

$NET_MLOAD_DIRECTORY = 827
$NET_MLOAD_MODULE_LOAD = 828
$NET_MODEM_INFO = 832
$NET_MODEM_SET = 833
$NET_MODEM_TYPE = 834
$ANALOG_SM_RESET = 848
$ANALOG_SM_CHANNEL = 849
$ANALOG_SM_INFO = 850
$AV_CONTROL_INFO = 864
$AV_CONTROL_SET = 865
$AV_CONTROL_WRT = 866
$AV_CONTROL_SEND = 867
$AV_CONTROL_DEV_LIST = 868
$OSD_ANALOG_INFO = 880
$OSD_ANALOG_SET_CONF = 881
$OSD_ANALOG_GET_CONF = 882
$OSD_ANALOG_SET_VOL = 883
$OSD_ANALOG_SET_TIME = 884
$OSD_ANALOG_SET_TEXT = 885
$OSD_ANALOG_DEL_TEXT = 886
$OSD_ANALOG_DISPLAY = 887
$NET_TUNER_FREQ_INFO = 896
$NET_TUNER_DEMOD_INFO = 897
$NET_TUNER_SET_SYMBOL = 898
$NET_TUNER_GET_SYMBOL = 899
$NET_TUNER_AGC_READ = 900
$NET_TUNER_BER_READ = 901
$NET_TUNER_START_SCANNING = 902
$NET_TUNER_NEXT_SCANNING = 903
$NET_TUNER_STOP_SCANNING = 904
$NET_TUNER_TUNING_INFO = 905
$NET_TUNER_TUNING_SET = 912
$NET_TUNER_SET_DUAL_SWITCH = 922
$NET_TUNER_GET_DUAL_SWITCH = 923
$MAC_MULTICAST_LIST = 928
$I2C_CI_INIT_COM = 944
$I2C_CI_INFO_CONNECT = 945
$I2C_CI_SEND_SPDU = 946
$DESCR_INFO = 960
$DESCR_PID_INFO = 961
$DESCR_START = 962
$DESCR_MODIFY = 963
$DESCR_STOP = 964
$DESCR_CLEAR = 965
$MODEM_BIS_SET = 976
$MODEM_BIS_DIAL = 977
$MODEM_BIS_SEND = 978
$MODEM_BIS_DECONNEXION = 979

$WIRELESS_INFO = 992
$WIRELESS_INIT = 993
$WIRELESS_SETUP = 994
$WIRELESS_SET_CONTROL = 995
$WIRELESS_SEND = 996
$WIRELESS_STOP = 997
$USB_DEVICE_INFO = 1008
$USB_GET_DESCRIPT = 1009
$USB_SET_DESCRIPT = 1010
$USB_SET_CONFIG = 1011
$USB_GET_CONFIG = 1012
$USB_GET_INTERF = 1013
$USB_GET_INTERF = 1014
$USB_SET_FEATURE = 1015
$USB_GET_STATUS = 1016
$USB_REQUEST = 1017
$USB_SEND_CTRL = 1018
$USB_SEND_INTERUP = 1019
$USB_RECV_INTERUP = 1020
$USB_SEND_BULK = 1021
$USB_RECV_BULK = 1022
$USB_GET_FRAME_NB = 1023
$USB_SEND_ISOCHRO = 1024
$USB_RECV_ISOCHRO = 1025
$USB_STOP = 1026
$USB_ABORT_PIPE = 1027
$USB_RESET_PIPE = 1028
$USB_CLEAR_PIPE = 1029
$USB_HALT_PIPE = 1030
$USB_SEND_HOST = 1031
$USB_RECV_HOST = 1032
$ARIB_SUBTITLE_INFO = 1040
$ARIB_SUBTITLE_SET_CONFIG = 1041
$ARIB_SUBTITLE_GET_CONFIG = 1042
$ARIB_SUBTITLE_START = 1043
$ARIB_SUBTITLE_PAUSE = 1044
$ARIB_SUBTITLE_RESUME = 1045
$ARIB_SUBTITLE_STOP = 1046
$ARIB_SET_RESOLUTION = 1047
$ARIB_GET_RESOLUTION = 1048
$DISK_INFO = 1056
$DISK_RESET = 1057
$DISK_POWER_CHANGE = 1058
$DISK_POWER_INFO = 1059
$DISK_SET_SECURITY = 1060
$DISK_DISABLE_SECURITY = 1061
$DISK_UNLOCK = 1062
$DISK_SET_AV_FEATURE = 1063

$DISK_GET_AV_FEATURE = 1064
$DISK_COMMAND = 1065
$CW_INFO = 1072
$CW_SELECT = 1073
$CW_SET_DESCR = 1074
$CW_LOAD_DESCR = 1075
$CW_FLUSH_DESCR = 1076
$CW_DESCR_LIST = 1077
$CW_PID_LIST = 1078
$CW_DEL_DESCR = 1079
$CW_DEL_ALL_DESCR = 1080
$CW_CHNL_FROM_PID = 1081
$CODE_LOADER_SECURE_INFO = 1088
$CODE_LOADER_SET_IMAGE = 1089
$CODE_LOADER_GET_TABLE = 1090
$CODE_LOADER_SET_RLOADER = 1091
$CODE_LOADER_GET_RLOADER = 1092
$CODE_LOADER_SET_PICTURE = 1093
$CODE_LOADER_GET_PICTURE = 1094
$CODE_LOADER_SET_INFO = 1095
$CODE_LOADER_GET_INFO = 1096
$PROXY_DLI_REPORT = 1104
$PROXY_DLI_INFO = 1105
$PROXY_SEND = 1106
$PROXY_CALL = 1107
$PROXY_IO = 1108
$PROXY_DLI_EVENT = 1109

Device ID

$dev_backup = 1
$dev_clock = 2
$dev_emm = 3
$dev_fdload = 4
$dev_lcard = 5
$dev_mload = 6
$dev_modem = 7
$dev_parallel = 8
$dev_power = 9
$dev_rcard = 10
$dev_sctv = 11
$dev_scvcr = 12
$dev_serial = 13
$dev_service = 14
$dev_sound_audio = 15
$dev_tuner = 16
$dev_display = 17
$dev_modem_bis = 18
$dev_nodify = 19

$dev_scaux = 20
$dev_remod = 21
$dev_unimodem = 22
$dev_mcom = 23
$dev_bus_1394 = 24
$dev_ts_remux = 25
$dev_ca = 26
$dev_keyboard = 27
$dev_mouse = 28
$dev_loader = 31
$dev_network = 32
$dev_pointer = 33
$dev_demux = 34
$dev_picture = 36
$dev_video = 37
$dev_subtitle = 38
$dev_irda = 39
$dev_pcmcia = 40
$dev_timer = 41
$dev_graphic = 42
$dev_dvb_subtitle = 43
$dev_dvb_ci = 44
$dev_net_mload = 45
$dev_net_modem = 46
$dev_analog_sm = 47
$dev_av_control = 48
$dev_osd_analog = 49
$dev_net_tuner = 50
$dev_mac = 51
$dev_i2c_ci = 52
$dev_descr = 53
$dev_mass_storage = 54
$dev_wireless = 55
$dev_usb = 56
$dev_arib_subtitle = 57
$dev_disk = 58
$dev_cw = 59
$dev_code_loader = 60
$dev_proxy = 61

Display Mode

$DISP_MODE_HIDE = 64
$DISP_MODE_CONTINUE = 512
$DISP_MODE_EXCLUSIVE = 8192

Enabled State

$OFF = 0

$ON = 1

Error ID

$E_FAILED = -1
$E_ALARM_TIME = 8192
$E_BAD_ECM = 8193
$E_DMUX_CHANNEL = 8194
$E_GRP_BUSY = 8195
$E_LCARD_KO = 8196
$E_NB_ALARM = 8197
$E_NAME_UNKNOWN = 8198
$E_NOT_DONE = 8199
$E_NO_ALARM = 8200
$E_NO_BUFFER = 8201
$E_NO_HARD_FILTER = 8202
$E_NO_LCARD = 8203
$E_NO_RCARD = 8204
$E_RCARD_KO = 8205
$E_RESSOURCE_DEVICE = 8206
$E_TIME_OUT = 8207
$E_ONLINE = 8208
$E_BAD_VALUE = 8209
$E_USER_CODE = 8210
$E_NO_DATA = 8211
$E_CALL_FAILED = 8212
$E_NO_DESCRAMBLER = 8213
$BAD_ADR_MEMORY = 8214
$E_NOT_ENOUGH_MEM = 8215
$E_REFUSED = 8216
$E_PULSE_NOT_SUPPORTED = 8231
$E_NO_MODEM = 8232
$E_FORBIDEN_STATE = 8240
$E_ANALOG_CARRIER = 8241
$E_TIME_OUT_CRC32 = 8242
$E_NO_CA = 8243
$E_PARAM_INIT = 8244
$E_BUFFER_FULL = 8245
$E_NACK = 8246
$E_NO_MATCH = 8247
$E_INTERNAL_ERROR = 8248
$E_EXTERNAL_ERROR = 8249
$E_CRC32 = 8250
$E_TRANSMISSION_ERROR = 8251
$E_SYM_RATE_FAILED = 8252
$E_BUS_RESET = 8253
$E_MAX_CHANNEL = 8254
$E_MAX_RATE = 8255
$E_NO_CHANNEL = 8256

$E_NO_PERIPH = 8257
$E_MAX_CONNECT = 8258
$E_NO_CONNECT = 8259
$E_RETRY_LIMIT = 8260
$E_BUS_OCCUPANCY_VIOLATION = 8261
$E_DUPLICATED_CHANNEL = 8262
$E_UNKNOWN_TRANS_CODE = 8263
$E_UNEXPECTED_CHANNEL = 8264
$E_HEADER_CRC_ERROR = 8265
$E_UNSOLICITED_RESPONSE = 8266
$E_REQUEST_DATA_ERROR = 8267
$E_RESPONSE_ACK_MISSING = 8268
$E_RESPONSE_DATA_ERROR = 8269
$E_RESPONSE_FORMAT_ERROR = 8270
$E_RESPONSE_RETRY_FAILED = 8271
$E_CYCLE_TOO_LONG = 8272
$E_BUS_RESET_START = 8273
$E_BUS_RESET_COMPLETED = 8274
$E_CABLE_POWER_FAIL = 8275
$E_BUS_LOOP_DETECTION = 8276
$E_BUS_FATAL_ERROR = 8277
$E_RMUX_CHANNEL = 8278
$E_DATA_ERROR = 8279
$E_ACKNOWLEDGE_MISSING = 8280
$E_MODEM_BUSY = 8281
$E_SERIAL_BUSY = 8282
$E_TUNING_PARAM = 8283
$E_PID_PARAM = 8284
$E_BAD_KEY = 8285
$E_BAD_SGN = 8286
$E_BAD_CIPH = 8287
$E_NO_HVN = 8288
$E_CONSISTENCY_INCONSISTENCY = 8289
$E_FLASH_ERR = 8290
$E_BAD_NB = 8291
$E_BAD_HANDLE = 8292
$E_ALREADY_RUNNING = 8293
$E_ADR_INUSE = 8294
$E_BAD_FORMAT = 8295
$E_MEMORY = 8296
$E_ALREADY_EXIST = 8297
$E_TABLE_FULL = 8298
$E_LOADER_UNKNOWN = 8299
$E_SET_UNKNOWN = 8300
$E_FIELD_UNKNOWN = 8301
$E_MAX_ECM = 8302
$E_CHIP_BUSY = 8303
$E_NO_CLUT = 8304

$E_NO_REGION = 8305
$E_PARALLEL_BUSY = 8306
$E_INVALID_FORMAT = 8308
$E_MAX_IMAGE = 8309
$E_IMAGE_NOT_FOUND = 8310
$E_MAX_TIMER = 8311
$E_EXIST = 8312
$E_DISPLAY_ERROR = 8313
$E_NO_TUNING = 8314
$E_MAX_DECOMP = 8315
$E_MAX_CONVERT = 8316
$E_NO_ANSWER = 8317
$E_HOST_NOT_FOUND = 8318
$E_NO_DNS = 8319
$E_MAX_ES = 8320
$E_MAX_CW = 8321
$E_NO_CW = 8322
$E_MAX_AREA = 8323
$E_BAD_CONNECT = 8324
$E_HARD_ALLOC = 8325
$E_BUSY = 8326
$E_MAX_REQUEST = 8327
$E_HDISK_ERR = 8329
$E_FIFO_USE = 8330
$E_PIPE_HALTED = 8331
$E_PERIPH_UNKNOWN = 0x0100
$E_CLIENT_MAX = 0x0101
$E_RESSOURCE_BUFFER = 0x0102
$E_SIZE_BUFFER = 0x0103
$E_BUFFER_USED = 0x0104
$E_RESSOURCE_EVT = 0x0105
$E_PRIORITE_UNKNOWN = 0x0106
$E_EVT_SRC_USED = 0x010B
$E_BAD_EVT_IO = 0x010C
$E_CMDE_UNKNOWN = 0x010D
$E_REPORT_PERIPH = 0x010E
$E_CLIENT_UNKNOWN = 0x010F
$E_EVENT_UNDECLARED = 0x0110
$E_NO_OUTLINING = 0x0111
$E_NO_MEMORY = 0x0112
$E_BAD_POOL = 0x0113
$E_BAD_PARAMETER = 0x0114
$E_BAD_SIZE = 0x011F
$E_OVERLAP = 0x0120
$E_OVERFLOW = 0x0300
$E_UNDERFLOW = 0x0301
$E_TIME = 0x0302
$W_LENGTH = 0x0303

Event Enabled State

$Enable = 0
$Disable_1 = 1

Event ID

$code_all = -1
$code_none = 0
$Ev_Timer = 14
$Ev_Module_End = 21
$EV_EMM_RECV = 32
$EV_EMM_21 = 33
$EV_EMM_22 = 34
$EV_LCARD_EXTRACT = 39
$EV_LCARD_INSERT = 40
$EV_LCARD_RESET = 41
$EV_RCARD_EXTRACT = 42
$EV_RCARD_INSERT = 43
$EV_RCARD_RESET = 44
$EV_MODEM_ENQ = 45
$EV_MODEM_OFF = 46
$EV_MODEM_MSG = 47
$EV_MODEM_CHAR = 48
$EV_PARALLEL_OFF = 49
$EV_PARALLEL_RECEIVE = 50
$EV_SERVICE_STATUS = 51
$EV_SERVICE_ON = 52
$EV_SCVCR_COMMUT = 53
$EV_SERIAL_OFF = 54
$EV_SERIAL_RECEIVE = 55
$EV_TUNER_BER = 56
$EV_TUNER_CARRIER = 57
$EV_SERVICE_ECM = 58
$EV_SERVICE_OFF = 59
$EV_SERVICE_GETALL = 60
$EV_SERVICE_BAD_NB = 61
$EV_SERVICE_FORMAT = 62
$EV_CA_DESCR = 63
$EV_MLOAD_GETALL = 64
$EV_MODEM_BIS_MSG = 65
$EV_MODEM_BIS_OFF = 66
$EV_POWER_RUNNING = 67
$EV_SERVICE_SYNCHRO = 68
$EV_SCAUX_COMMUT = 69
$EV_CLOCK_TIME = 70
$EV_UNIMODEM_RCV = 71

$EV_UNIMODEM_OFF = 72
$EV_CA_APP_EMM = 73
$EV_CA_EMM = 74
$EV_CA_EMM_ERROR = 75
$EV_CA_EMM_SM = 76
$EV_CA_ECM_SM = 77
$EV_CA_GETALL = 78
$EV_CA_ECM_MANAGER = 79
$EV_TUNER_AUDIO_DUAL_MODE = 80
$EV_MCOM_RCV = 81
$EV_MCOM_DATAGRAM = 82
$EV_MCOM_ERROR = 83
$EV_MCOM_OFF = 84
$EV_CA_APP_ECM = 85
$EV_I2C_CI_SPDU_RCV = 86
$EV_I2C_CI_TC = 87
$EV_MAC_MULTICAST_REMOVE = 88
$EV_MAC_MULTICAST_ADD = 89
$EV_ANALOG_SM_CHANGE = 90
$EV_SUBTITLE_ANALOG = 91
$EV_NET_TUNER_CARRIER = 92
$EV_NET_TUNER_BER = 93
$EV_NET_TUNER_DUAL_MODE = 94
$EV_I2C_CI_CONNECT = 95
$EV_BUS_1394_RCV_FCP = 96
$EV_BUS_1394_CHANNEL = 97
$EV_BUS_1394_CONFIG = 98
$EV_BUS_1394_CONNECT = 99
$EV_BUS_1394_LO_EVENTS = 100
$EV_BUS_1394_OFF = 101
$EV_BUS_1394_HI_EVENTS = 102
$EV_UNIMODEM_SUSP = 103
$EV_68 = 104
$EV_69 = 105
$EV_ARIB_SUBTITLE_INFO = 106
$EV_VIDEO_FORMAT_HD = 107
$EV_VIDEO_BUFFERS_LIST = 108
$EV_DESCR_STATUS = 109
$EV_I2C_CI_OFF = 110
$EV_I2C_CI_ERROR = 111
$EV_TS_REMUX_ECM = 112
$EV_TS_REMUX_GETALL = 113
$EV_TS_REMUX_BAD_NB = 114
$EV_TS_REMUX_STREAM = 115
$EV_WIRELESS_RCV = 116
$EV_WIRELESS_OFF = 117
$EV_USB_ATTACH = 118
$EV_USB_DETACH = 119

$EV_SUBTITLE_BUFFERS_LIST = 120
$EV_DVB_SUBTITLE_ERROR = 121
$EV_DVB_SUBTITLE_BUFFERS_LIST = 122
$EV_VIDEO_DISPLAY = 123
$EV_DVB_CI_SPDU_RCV = 124
$EV_DVB_CI_CONNECT = 125
$EV_DVB_CI_ERROR = 126
$EV_DVB_CI_OFF = 127
$Ev_keypress = 128
$EV_RESERVED_81 = 129
$EV_RESERVED_82 = 130
$EV_RESERVED_83 = 131
$Ev_frontalpress = 132
$EV_RESERVED_85 = 133
$EV_RESERVED_86 = 134
$Ev_Module_Open_Channel = 135
$EV_RESERVED_88 = 136
$EV_RESERVED_89 = 137
$EV_RESERVED_8A = 138
$EV_RESERVED_8B = 139
$EV_RESERVED_8C = 140
$EV_RESERVED_8D = 141
$EV_RESERVED_8E = 142
$EV_RESERVED_8F = 143
$EV_RESERVED_90 = 144
$EV_RESERVED_91 = 145
$EV_RESERVED_92 = 146
$EV_RESERVED_93 = 147
$EV_RESERVED_94 = 148
$EV_RESERVED_95 = 149
$EV_MASS_STORAGE_BUFFER_LIST = 150
$EV_97 = 151
$EV_VIDEO_FORMAT = 152
$EV_PICTURE_BUFFERS_LIST = 153
$EV_GRAPHIC_BUFFERS_LIST = 154
$EV_POINTER_AREA_IN = 155
$EV_POINTER_AREA_OUT = 156
$EV_POINTER_MOUSE_MOTION = 157
$EV_POINTER_BUTTON_PRESS = 158
$EV_POINTER_BUTTON_RELEASE = 159
$EV_LCARD_RECEIVE = 160
$EV_LCARD_OFF = 161
$EV_RCARD_RECEIVE = 162
$EV_RCARD_OFF = 163
$EV_LCARD_PPS = 164
$EV_RCARD_PPS = 165
$EV_DISK_POWER_CHANGE = 166
$EV_CW_DESCR_OFF = 167

$EV_SCTV_DSWITCH = 168
$EV_SCTV_HDINFO = 169
$EV_TUNER_MULTI_TS = 170
$EV_USB_CONF_HOST = 171
$EV_USB_RECV_HOST = 172
$EV_DISK_OUT_OF_ORDER = 173
$EV_PROXY_DLI_REQUEST = 174
$EV_PROXY_RECEIVE = 175
$EV_PROXY_B0 = 176
$EV_PROXY_B1 = 177
$EV_PROXY_B2 = 178
$EV_PROXY_B3 = 179
$EV_PROXY_B4 = 180
$EV_PROXY_B5 = 181
$EV_PROXY_B6 = 182
$EV_PROXY_B7 = 183
$EV_PROXY_B8 = 184
$EV_PROXY_B9 = 185
$EV_PROXY_BA = 186
$EV_PROXY_BB = 187
$EV_PROXY_BC = 188
$EV_PROXY_BD = 189
$EV_PROXY_BE = 190
$EV_PROXY_BF = 191
$EV_NET_MODEM_UCC = 224
$EV_POD_EXT_CHANNEL_RCV = 245
$EV_F6 = 246
$EV_NET_STACK_ICMP = 247
$EV_NET_MODEM_LOGOUT = 248
$EV_NET_MODEM_LOGIN = 249
$EV_NET_STACK_PP_STATE = 250
$EV_DVB_CI_TC = 251
$EV_NET_PPP_AUTHENTICATE = 252
$EV_CA_SM_MSG = 253
$EV_AUDIO_BUFFERS_LIST = 254
$EV_SOUND_AUDIO_TYPE = 255
$EV_PMT_RECV = 512
$EV_PAT_RECV = 520
$EV_CAT_RECV = 521
$EV_LCARD_RESET = 678
$EV_ECM_RECV = 683
$EV_LCARD_INPUT = 700

File Mode/Type

$MODE_BINARY = 0
$MODE_TEXT = 4

Justification Constant

$LEFT = 0
$CENTER = 1
$RIGHT = 2

Key ID

$KEY_0_AV = 601
$KEY_1 = 602
$KEY_2 = 603
$KEY_3 = 604
$KEY_4 = 605
$KEY_5 = 606
$KEY_6 = 607
$KEY_7 = 608
$KEY_8 = 609
$KEY_9 = 610
$KEY_A = 611
$KEY_B = 612
$KEY_C = 613
$KEY_D = 614
$KEY_E = 615
$KEY_Pilot = 616
$KEY_Guide = 617
$KEY_Fav = 618
$KEY_Exit = 619
$KEY_Pers = 620
$KEY_Plus = 621
$KEY_Serv = 622
$KEY_TvSat = 623
$KEY_Mute = 624
$KEY_OnOff = 625
$KEY_Up = 626
$KEY_Down = 627
$KEY_Left = 628
$KEY_Right = 629
$KEY_Ok = 630
$KEY_Mail = 638
$KEY_PgUp = 65534
$KEY_PgDown = 65535

Lock Flag Constant

$Lock_OFF = 0
$Lock_ON = 1

Memory Attribute ID

$BYTE_ORDER = 0

$STRING_FORMAT = 1

Panel Attribute ID

$INPUT = 0
$WIDGETS_ACTIVE_ONLY_GROUP_GE = 1
$OPTION_2 = 2
$WIDGETS_SET_GROUP = 3
$OPTION_4 = 4

Panel Attribute Value

$BLOCKED = 0
$REACTIVATE = 1

Panel Key ID

$Key_Panel_OK = 2
$Key_Panel_ESC = 3
$Key_Panel_UP = 4
$Key_Panel_DOWN = 5

Position Panel Reference Code

$PANEL_POS_PANEL_TOP_LEFT = 0
$PANEL_POS_PANEL_CENTER = 1
$PANEL_POS_DISPLAY_TOP_LEFT = 2
$PANEL_POS_DISPLAY_TOP_RIGHT = 3
$PANEL_POS_DISPLAY_BOTTOM_LEFT = 4
$PANEL_POS_DISPLAY_BOTTOM_RIGHT = 5
$PANEL_POS_DISPLAY_CENTER = 6
$PANEL_POS_PANEL_TOP_LEFT_07 = 7
$PANEL_POS_PANEL_CENTER_08 = 8
$PANEL_POS_DISPLAY_UNDER_CENTER = 9
$PANEL_POS_DISPLAY_TOP_LEFT_0A = 10
$PANEL_POS_UNKNOWN_0B = 11
$PANEL_POS_UNKNOWN_0C = 12
$PANEL_POS_PANEL_TOP_LEFT_0D = 13
$PANEL_POS_PANEL_TOP_LEFT_0E = 14
$PANEL_POS_PANEL_TOP_LEFT_0F = 15

Quit Panel Enabled

$Quit_Panel_OFF = 0
$Quit_Panel_ON = 1

Seek Mode Constant

$SEEK_SET = 0
$SEEK_CUR = 1
$SEEK_END = 2

Slider Action Constant

$SLIDER_INIT = 0
$SLIDER_SET = 1
$SLIDER_STEP = 2

String Type Constant

$RAW_STRING = 0
$C_STRING = 1
$L1_STRING = 2
$L2_STRING = 3

TCS Channel Flag Command

$OptionRead = 0
$OptionWrite = 1

Timer Operating Mode

$TIMER = 0
$CLOCK = 1

Warning ID

$W_EXIST = 8448
$W_LOAD_UNKNOWN = 8449
$W_NAME_UNKNOWN = 8450
$W_NO_DATA = 8451
$W_NO_SAMPLE_PLAY = 8452
$W_SECTION_DONE = 8453
$W_SECT_UNKNOWN = 8454
$W_LOAD_DONE = 8455
$W_MPEG_CHIP_BUSY = 8456
$W_NO_PICTURE = 8457
$W_MAX_PICTURE = 8458
$W_SYM_RATE_FAILED = 8459
$W_DONE = 8464
$W_APP_REQ = 8465
$W_NO_UPDATE = 8466
$W_SOCKET_DOWN = 8467
$W_NOT_HANDLE = 8468
$W_MAX_IMAGE = 8469
$W_NO_IMAGE = 8470
$W_DECOMP_NEW_PIXMAP = 8471
$W_CRC_32 = 8472
$W_DECOMP_NEW_IMAGE = 8473
$W_USER_REQ = 8474
$W_DECOMP_NEW_TEXT = 8475
$W_NOT_ENOUGH_DATA = 8476
$W_NOT_ENOUGH_FRAGMENT = 8477

Window Attribute

$INK = 0
$PAPER = 1
$BORDER_WIDTH = 6