Title: BROADCAST DOWNLOAD TECHNIQUE

(57) Abstract: In a wireless microcell distribution system, a multicast system is provided for broadcasting software updates, enable and disable capabilities, channel reassignment or other downloadable data simultaneously to a plurality of cable microcell integrators, thereby precluding the lengthy unicast process of updating cable microcell integrators manually or suffering prior unreliable message delivery in which each of the cable microcell integrators in individually addressed and the response tallied. In one embodiment, the request is transmitted simultaneously in a broadcast mode to multiple cable microcell integrators with no response required. Thereafter, each cable microcell integrator is queried on a unicast basis as to receipt of broadcast information and as to whether or not any portion must be re-broadcast. If so optionally the entire information set can be rebroadcast, with only affected cable microcell integrators downloading the broadcast information. The multicast broadcast technique thus provides rapid simultaneous updating of each of the cable microcell integrators from a central location.
Published:

- Without international search report and to be republished upon receipt of that report.

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.
TITLE OF INVENTION

BROADCAST DOWNLOAD TECHNIQUE

FIELD OF INVENTION

This invention relates to wireless microcell distribution systems and more particularly to a technique for providing software updates and other information to a plurality of microcells for decreasing the amount of download time required.

BACKGROUND OF THE INVENTION

In wireless microcell distribution systems, a number of microcells are coupled to a cable for the transmission and receipt of wireless signals to and from wireless terminals. It will be appreciated that there may be as many as thirty six microcells coupled to a cable, and that each of these microcells on occasion must be provided with downloadable software updates. The type of information which is transmitted to each of the microcells is for instance, software to reconfigure each of the microcells, to reconfigure the enable and disable capabilities of each of the microcells, to change the channel assignments, or to provide other information to each of the microcells.

In the past, this has been accomplished on a unicast basis in which messages from a head end control unit are transmitted one at a time to each of the microcells, with each of the microcells having a so-called cable microcell integrator thereat. The cable microcell integrator responds to a request message from the head end interface converter and reconfigures its own internal software to comply with the information transmitted to it. Upon the download, the cable microcell integrator transmits a response back to the head end control unit indicating the status of the downloaded information.

It will be appreciated that this unicast technique requires considerable time. For instance, it may require as much as an hour to download 1,000 messages to a particular cable microcell
integrator. This is multiplied by the number of cable microcell integrators communicating with the particular head end control unit, such that a software download can sometimes take up to a number of weeks to accomplish.

While it will be appreciated that the prior techniques adequately permit downloading of information to each of the cable microcell integrators, it is preferable to be able to accomplish the process in a shorter period of time.

**SUMMARY OF THE INVENTION**

Rather than addressing each of the cable microcell integrators separately, in an improved process, a message is sent from the head end control unit to each of the cable microcell integrators to reconfigure each cable microcell integrator to go into the broadcast mode. This transmission is done on a unicast basis.

Thereafter, the downloadable information is broadcast to all of the cable microcell integrators in a so-called broadcast mode in which the information is transmitted simultaneously to each of the cable microcell integrators.

After the broadcast has been completed, an exit broadcast mode message is broadcast to each of the cable microcell integrators. Thereafter, in a unicast mode, each of the cable microcell integrators is queried as to the broadcast status. In one embodiment, what each of the cable microcell integrator computes is a check sum, indicative of the confirmed receipt of each of the downloads.

Upon ascertaining at each of the cable microcell integrators the status of each broadcast, this information is transmitted back to the head end control unit on a unicast basis. If it appears that a large portion of the message has been lost or could have been misinterpreted by a number of cable microcell integrators, in one embodiment the head end control unit re-broadcasts either all or part
of the download with those cable microcell integrators having confirmed receipt of the original message simply ignoring the additional broadcast or that portion of it that they have properly received.

In this manner, the process which in some cases can take as much as several weeks, is completed in a matter of hours or even minutes. It is important to note that only that information which is required of each cable microcell integrator which it did not get, is required to be transmitted back to it.

Thus, the subject system combines unicast and broadcast in an efficient manner to be able to provide detection and correction of erroneous messages at each cable microcell integrator.

In summary, in a wireless microcell distribution system, a multicast system is provided for broadcasting software updates, enable and disable capabilities, channel reassignments or other downloadable data simultaneously to a plurality of cable microcell integrators, thereby precluding the lengthy unicast process of updating cable microcell integrators manually or suffering prior unreliable message delivery in which each of the cable microcell integrators is individually addressed and the response tallied. In one embodiment, the request is transmitted simultaneously in a broadcast mode to multiple cable microcell integrators with no response required. Thereafter, each cable microcell integrator is queried on a unicast basis as to receipt of broadcast information and as to whether or not any portion must be re-broadcast. If so optionally, all or portions of the information set can be re-broadcast, with only affected cable microcell integrators downloading the broadcast information. The multicast broadcast technique thus provides rapid simultaneous updating of each of the cable microcell integrators from a central location.
BRIEF DESCRIPTION OF THE DRAWINGS

These and other features of the Subject Invention will be better understood in connection with the Detailed Description in conjunction with the Drawings of which:

Figure 1 is a block diagram of a prior art unicast system in which information is downloaded to each of the cable microcell integrators separately, with the response from each of the cable microcell integrators also being evaluated on a microcell-by-microcell basis;

Figure 2 is a block diagram of the subject system in which software updates and other information are broadcast simultaneously to each of the cable microcell integrators with no response required;

Figure 3 is a block diagram of the system in Figure 2 in which after the information has been broadcast to each of the cable microcell integrators, a query message is sent on a unicast basis to each of the cable microcell integrators and the response therefrom evaluated; and

Figure 4 is a flow chart illustrating the sequence of events in the download of information to be broadcast to each of the cable microcell integrators.

DETAILED DESCRIPTION

Referring now to Figure 1, in a wireless microcell distribution system, a head end control unit 10 is coupled to a head end interface converter 12 which is serially connected to a number of cable microcell integrators 14, 16, 18 and 20 over a network which includes a cable television plant, be it fiber optic, cable or microcell. It will be appreciated that from time to time, each of the cable microcell integrators must be provided with software updates, channel reassignments or enable and disable capabilities which must be changed. Thus on occasion requirements include fixing, software bugs, enhancing capabilities or channel configurations. In the past, as mentioned before, these software updates have been accomplished by downloading to each of the cable microcell...
integrators, the required information on a separate basis. Thus for instance, each of the cable
microcell integrators is addressed separately and is provided with the downloaded information at
that time. Thereafter, validation of receipt of such information is transmitted back to the head end
control unit over the network. Should some of the information not be properly downloaded, the
proper information is resent to the particular cable microcell integrator requiring it.

Referring now to Figure 2, in contradistinction to the above method of providing software
updates to the cable microcell integrators, in the subject system the information is transmitted to
each of the cable microcell integrators on a multicast basis in which the information to be
downloaded is transmitted simultaneously to each of the cable microcell integrators. This is
indicated by information on a request path here illustrated at 22.

Referring now to Figure 3, while the broadcast of the downloadable information is
accomplished in a multicast or simultaneous mode, after the information has been transmitted, the
system is placed in a unicast query mode, as illustrated in Figure 3, in which a query message, as
illustrated in line 24, is sent to a specific cable microcell integrator and the response therefrom
evaluated, with the response being sent back as illustrated on path 26.

In one embodiment, the query message requests a response from the addressed cable
microcell integrator for a validation message, in one case a simple check sum. Other types of
validation messages are possible, including those that indicate which portion of which message has
not been properly downloaded or received. As is typical in such validation techniques, check sums
are provided on sections of the downloaded information. Another technique for validation includes
a cyclic redundancy check.

Referring now to Figure 4, from the point of view of the flow of information, the head end
control unit, as illustrated in box 30, transmits to each cable microcell integrator the need to place
the cable microcell integrator in its broadcast mode. Thereafter, the downloadable information is broadcast, as illustrated at box 32, to send N broadcast messages, which in some cases can exceed 1,000 messages. These messages, as indicated above, are sent simultaneously to each of the cable microcell integrators from the particular head end control unit.

As can be seen from box 34, after the broadcast of the information to each of the cable microcell integrators, an “exit” broadcast message is simultaneously multicast to each of the cable microcell integrators followed by a unicast “query” message as illustrated at box 36 as to the broadcast status. Thus in this task, each of the cable microcell integrators is serially queried by the head end control unit, with its broadcast status evaluated one at a time. Thereafter, assuming that there is a common transmission of any, all or a part of a broadcast message, the original information may be broadcast back to all of the cable microcell integrators as indicated by dotted line 38, with each of the cable microcell integrators responding only to that portion of the message that was not properly received.

What follows is a program written in C, describing the combined unicast-broadcast system for one embodiment of the subject invention.

Code Pertaining to CDL Extracted from HLROM.C

/*
 * TITLE: Proc_Mess
 * DESCRIPTION: To process the messages received while in PROM and some of the messages
 * received while in EEPROM will call this procedure.
 * INPUTS: none
 * OUTPUTS: continue_Download_Flag: True or False
 * ASSUMPTIONS/LIMITATIONS:
 * This routine handles broadcast messages
 * This procedure does not handle every CMI message.
 * This procedure will NACK messages and report a reason for the NACK
 * Revision History:
 * PCSC-122 3/15/98
 * This procedure now sends the response msg to the appropriate source
 * CMI Number of BROADCAST indicates that the message received was a broadcast
 * message and therefore should not be acknowledged by the CMI.
 * dl_stat is now a local variable not a global.
 * This procedure now processes new download messages.
 */
This procedure now checks for multiple faults, not just two.

Removed the xdata declaration on local variable j.

This routine can now receive broadcast messages; but this particular
CMI may not want to be processing the broadcast message: check the
global flag prior to processing.

This procedure now checks the global flag download_mode prior to
processing any download messages.

This procedure now handles filling all/part of EEPROM with msg
This procedure now handles performing a checksum on any defined area

PCSC-161 Added the ACT_CMI message to ROM - note it does not perform some of
the action if the CMI is in ROM waiting for download

4/24/98 Set EEPROM to false when CDL xsum is received.

4/28/98 This procedure will handle a random delay to the response of a broad-
cast RUA_LOST_CMI message.

PCSC-174 5/20/98 The flush routine must ensure download_mode is active to process !

PCSC-178 6/9/98 This routine now processes freq_stat messages.

This routine is set up to handle two future features on
the Rua_Lost_Cmi message.

PCSC-213 7/17/98 Changed the freq_stat_msg to report back the RAM value of
fwd_channel and type (not what is in EEPROM)

On ACT_CMI. if sector changes, re-tune FWD PLLs

/****************************************************************************
/* This is the Process Message Routine */
unsigned char Proc_Msg(void ) {

unsigned char i,j;
unsigned int int_i;
unsigned char xdata *addr;
unsigned int s_addr; /* will contain a pointer */
unsigned char pattern;
unsigned char load_stat;
unsigned char contin;
unsigned char ret_stat;
unsigned long chksum;
unsigned char *x_addr;
unsigned char cnt;
unsigned char n, temp;

contin = TRUE; /* unless otherwise changed by any proc. called from this proc */
recv_brdcast_msg = msg_in.dat raw.dat[0] 

recvd_brdcast_msg = msg_in.dat 0.dat[0] & BRCDST_MSG; /* BP#1 contains num/sector OR brdcast */

/****===========================================================================*/
/ * Ensure that any broadcast message received is to be processed by this CMI */
/****===========================================================================*/
if ( recv_brdcast_msg == BROADCAST ) /* was the msg received broadcast ? */
{ /* yes - check to see that the process_broadca_msg flag has been previously set */

if ( process_broadca_msg ) /* are we currently accepting brdcast ? */
{ /* NO - but, it may be a Broadcast call for Lost CMI ! check before ignoring */

if ( lost_cmi ) /* are we a lost_cmi ? */
{ /* yes - check to see if this is a lost inquiry */

if ( msg_in_number != RUA_LOST_CMI )
{ /* NO - it's from rs232 - we can process it */

contin = TRUE;

Drop_It();

} else /* it's from rs232 - we can process it */

contin = TRUE;

Drop_It();

} else /* it is an RUA_LOST_CMI msg - process it */

contin = FALSE; /* no, do not process and do not ack ! */

Drop_It();

} else /* not a lost CMI, and not processing Brdcast, drop it on the floor */

contin = FALSE; /* no, do not process and do not ack ! */

Drop_It();

}
else
{" /* the message received was not broadcast, but should we process it ? */

/* Since we are in ROM, we need to have an established communications with our
HIC in order to properly respond to any message received. Until we receive
an RUA_LOST_CMI (or ACT_CMI), we will not process any messages ! */

if ((lost_cmi) && (neuron_msg_flag)) /* are we a lost_cmi? AND receiving Neuron ? */
{/ /* yes - check to see if this is a lost inquery OR an Activate CMI */
  if ((msg_in.number == RUA_LOST_CMI ) || (msg_in.number == ACT_CMI_MSG ))
    contin = TRUE;
  /* process and do not ack */
  else /* no, do not process the message */
    contin = FALSE;
}
else /* we are not lost (or we are but msg is from 232) - so process the message */
    contin = TRUE;

if ( contin == TRUE )
{" /* this CMI is supposed to process the message */

switch( msg_in.number ) /* which message was received ? */
{/ /* improved download commands (post Version 4 EEPROM code) */
  /* case TIMER_DOWN_REQ */
  case CHIP_CDL_FILL_PATT /* indicates a fill pattern */
  |
  case CHIP_CDL_NORM /* all data in one record */
  |
  case CHIP_CDL_START_CONT /* start of a continuation load */
  |
  case CHIP_CDL_CONT /* a continuation load */
  |
  case CHIP_CDL_FUTURE /* a future enhancement download */
  |

  if ( download_mode == TRUE ) /* are we supposed to be getting these ? */
    { /* yes - process the download message */
      load_stat = Improved_DN_Code(); /* process the download msg */
      Return_Message(); /* acknowledge msg is processed */
    } else /* we are not in the correct mode to receive this message ! */

    Send_Back_Nack( /* Nack the msg is NOT processed */
      CDL_WRONG_MODE, /* Reason code */
      msg_in.number); /* supporting data */

    break;

/* case CHIP_CDL_FREQ_STAT_MSG: */ $$$$$$$ Read the Frequency status $$$$$$$ */

  msg_in.dat.send_freq.stat.rev_pri_freq = *(char xdata *)EEP_REV_PRI_FREQ;
  msg_in.dat.send_freq.stat.rev_div_freq = *(char xdata *)EEP_REV_DIV_FREQ;
  msg_in.dat.send_freq.stat.fwd_chn = fwd_chn;
  msg_in.dat.send_freq.stat.fwd_type = fwd_type;
  msg_in.dat.send_freq.stat.pcs_tx_freq_msb = *(char xdata *)EEP_PCS_CHAN_MSB;
  msg_in.dat.send_freq.stat.pcs_tx_freq_lsb = *(char xdata *)EEP_PCS_CHAN_LSB;

  Return_Message(); /* acknowledge msg is processed */

  break;

/* case CHIP_CDL_CALC_SECT_XSUM: */ $ CHIP calculate sector xsum data $ */

  Return_Message(); /* acknowledge msg is received prior: it takes 10 seconds ! */
  Calc_sector_xsum();

  break;

/* */
case CMI_CDL_RPT_SCT_XSUM: /* CMI Report sector xsum data */

"CASE CMI_CDL_RPT_SCT_XSUM: /* CMI Report sector xsum data */

/* msg_in.dat.rpt.xsum.fld2 = sector_xsum[0];
msg_in.dat.rpt.xsum.fld3 = sector_xsum[1];
msg_in.dat.rpt.xsum.fld4 = sector_xsum[2];
msg_in.dat.rpt.xsum.fld5 = sector_xsum[3];
msg_in.dat.rpt.xsum.fld6 = sector_xsum[4];
msg_in.dat.rpt.xsum.fld7 = sector_xsum[5];
msg_in.dat.rpt.xsum.fld8 = sector_xsum[6];
msg_in.dat.rpt.xsum.fld9 = sector_xsum[7];
msg_in.dat.rpt.xsum.fld10 = 0; /* clear for calc xsum */
for (i=0; i<8; i++) /* now clac xsum of msg */
msg_in.dat.rpt.xsum.fld10 += sector_xsum[i]; /* byte checksum of this msg */
Return_Message(); /* acknowledge msg is processed */
break;

/* CASE CMI_CDL_RESET: /* CMI Reset instructions */

/* CASE CMI_CDL_RESET: /* CMI Reset instructions */

/* first verify the patterns in the message are correct */
if (msg_in.dat.cdl_reset_cmi.patt1 == 0xB) &&
(msg_in.dat.cdl_reset_cmi.patt2 == 0xE) &&
(msg_in.dat.cdl_reset_cmi.patt3 == 0x2) &&
(msg_in.dat.cdl_reset_cmi.patt4 == 0xF) { /* correct pattern - so process the message */
Return_Message(); /* acknowledge msg is received */

/* allow a first time run of the new downloaded code to store its version to EE */
Write_EEPROM(char xdata *)EEP_CMI_ONLINE, FALSE); /* not acquired is the key ! */
Write_EEPROM(char xdata *)EEP_INTENT_RESET, TRUE);

/* update the flag in EE so the reset code knows where to get values from */
if (msg_in.dat.cdl_reset_cmi_source == 0xEE) /* the NIC wants reset to last known values ? */
/* yes - OR in the bit */
pattern = (char xdata *)EEP_CMI_STATUS | PU_FRM_STATUS_BIT;
Write_EEPROM(char xdata *)EEP_CMI_STATUS, pattern);

else if (msg_in.dat.cdl_reset_cmi_source == 0xDE) /* want reset from Defaults ? */
/* yes - Clear the bit */
pattern = (char xdata *)EEP_CMI_STATUS & ~PU_FRM_STATUS_BIT;
Write_EEPROM(char xdata *)EEP_CMI_STATUS, pattern);

/* don't update EE if the value isn't one of the above ! */
/* indicate the reset instructions */
load_rtrc = msg_in.dat.cdl_reset_cmi.reset_instr;

else /* the message is corrupted or illegal */
Send_Back_Hack(); /* Nack the msg is NOT processed */
ENCODING_MSG_CORRUPTED, /* reason code */
msg_in.number); /* support data */
break;

/* CASE CMI_CDL_STAT_INSTR: /* indicates download instructions */

/* CASE CMI_CDL_STAT_INSTR: /* indicates download instructions */

/* first verify the patterns in the message are correct */
if (msg_in.dat.cdl_stat_recv.patt1 == 0xB) &&
(msg_in.dat.cdl_stat_recv.patt2 == 0xE) &&
(msg_in.dat.cdl_stat_recv.patt3 == 0x2) &&
(msg_in.dat.cdl_stat_recv.patt4 == 0xF) { /* correct pattern - so process the message */
download_mode = msg_in.dat.cdl_stat_recv.dm_mode; /* indicate dm mode */
process_broadcastmsgs = msg_in.dat.cdl_stat_recv.proc_brdcast_msgs; /* indicates
brdcast instructions */
verify_page_write = msg_in.dat.cdl_stat_recv.verify_page_write; /* verify or not */
if (msg_in.dat.cdl_stat_recv.spare1)
"are we to invalidate last sector xsum bit map ? */
/* yes */
for (i=0; i<8; i++) /* now clac xsum of msg */
sector_xsum[i]=0xFF; /* invalidated old bit map of sector checksum */

Return_Message(); /* acknowledge msg is processed */
}
else /* the message is corrupted or illegal */
{
    Send_Back_Nack(  /* Nack the msg is NOT processed */
        ENCODED_MSG_CORRUPTED, /* reason code */
        msg_in.number);  /* support data */
}
break;

/* ========================================================= */
case CMD_CDL_FLUSH : /* indicates a 'flush' of the EEPROM */
/* ========================================================= */
    if ( download_mode == TRUE ) /* are we supposed to be getting these ? */
        { /* yes - process the message */
            /* first verify the patterns in the message are correct */
            if ( ( msg_in.dat.cdl_flush.patt1 == 0x0F )
                & ( msg_in.dat.cdl_flush.patt2 == 0xA )
                & ( msg_in.dat.cdl_flush.patt3 == 0xD )
                & ( msg_in.dat.cdl_flush.patt4 == 0xE ) )
            ( /* correct pattern - so process the message */

            /* NOTE: we are verifying these writes which does take longer - but since it */
            /* only takes one message from the HECU to clear all of EEPROM - who cares ! */
            s_addr = msg_in.dat.cdl_flush.s_addr; /* indicate the start address */
            count = msg_in.dat.cdl_flush.count;  /* indicate the amount of bytes */
            pattern = msg_in.dat.cdl_flush.pattern; /* indicates the pattern */
        }

    /* Ensure that we are not in a download (using the contiguous buffer) */
    if ( ! improved_di_struct.in_use )
        ( /* structure is not in use by a download - so we can use it in the flush */
            
        )
    if ( ! verify_page_write ) /* if we are not going to verify */
        Return_Message(); /* acknowledge msg is received prior to processing */

    for ( int_i=0; int_i< MAX_CDL_BUF; int_i++ ) /* put the pattern in a buffer */
        contiguous_di_data[int_i] = pattern;

    i = count / MAX_CDL_BUF;  /* number of full buffer writes */
    for( j=0; j<i; j++)/* using page_write put out as many MAX_CDL_BUF byte buffers */
        {
            Enable_EEPROM(); /* enable EEPROM for writes */
            ret_stat = Page_Write_EEPROM( /* function */
                (xdata unsigned int*)s_addr, /* dest. address */
                MAX_CDL_BUF,  /* num bytes to write */
                contiguous_di_data, /* pointer to source */
                verify_page_write ); /* verify flag */

            Protect_EEPROM(); /* protect the EE from writes */
            s_addr += MAX_CDL_BUF; /* move the destination pointer up */

        if ( ret_stat == SUCCESS )
            /* the page write failed - indicate failure to the HECU */
        Set_Up_A_Nack( PAGE_WRITE_FAILURE, /* reason */
            ret_stat); /* support data: the failure stat code */
        break; /* from loop no need to continue ! */
    }
}
/* end for loop */

if( ret_stat == SUCCESS )
    ( /* continue and put out the remainder of the count */
        i = count % MAX_CDL_BUF; /* the last (remainder) write */

        Enable_EEPROM(); /* enable EEPROM for writes */
        ret_stat = Page_Write_EEPROM( /* function */
            (xdata unsigned int*)s_addr, /* dest. address */
            contiguous_di_data, /* pointer to source */
            verify_page_write ); /* verify flag */

        Protect_EEPROM(); /* protect the EE from writes */
    )/* end if the ret_stat != Fail */

if ( ! verify_page_write ) /* if we are verifying */
    { /* then we did NOT respond prior to processing - so respond now */
        if ( recvd_brdcast_msg != BROADCAST )
            ( /* not a broadcasted message */


if ( ret_stat == SUCCESS )
    Send_Back_Nack(
        PAGE_WRITE_FAILURE, /* reason code */
        msg_in.number ); /* support data */
else /* the message processed was successful */
    Return_Message(); /* acknowledge msg is processed */
} /* end if the msg was a broadcasted msg */
else/* the structure we need to use is already in use - Nack the msgs */
{
    Send_Back_Nack(
        CDL_STRUCT_IN_USE, /* reason code */
        msg_in.number ); /* support data */
}
else /* the message is corrupted or illegal */
{
    Send_Back_Nack( /* Nack the msg is NOT processed */
        ENCODED_MSG_CORRUPTED, /* reason code */
        msg_in.number); /* support data */
}
else/* we are not in the correct mode to receive this message */
{
    Send_Back_Nack( /* Nack the msg is NOT processed */
        CDL_WRONG_MODE, /* Reason code */
        msg_in.number); /* supporting data */
}
break;
/* 
*Calculates a checksum of the code in EEPROM
*/
/* 
*/
x_addr = (unsigned char *)msg_in.dat.cdl_part_xsum.s_addr;
    /* Where to start */
chksum = 0;
    /* start at zero */
count = msg_in.dat.cdl_part_xsum.count;
for ( int_i = 0; int_i < count; int_i++)
{
    chksum += *x_addr; /* add in this byte */
    x_addr++;
        /* next byte */
}
(unsigned long)msg_in.dat.cdl_part_xsum.xsum = chksum; /* report it back */
Return_Message(); /* acknowledge msg is processed */
break;

/* 
* 
*/
/* 
* 
*/
/* 
* 
*/
/* 
* 
*/

/* 
* 
* 
* 
* 
*/

** TITLE: Improved_DL_Code
** DESCRIPTION:
This procedure is the improved download code. It processes a number of messages, and to reduce the amount of data sent if the data is redundant. The procedure also calls a Page Write EE function. This procedure can write data to the EEPROM much faster and as many bytes as needed (not hard coded to 6 bytes).
** INPUTS: None
** OUTPUTS: Continue Download flag: T or F - this is the end of the download
** ASSUMPTIONS/LIMITATIONS:
This procedure will look into the received msg for indications of size. The maximum amount of contiguos DL data is 256 bytes.
This procedure will not properly handle re-sends of any CONTINUATION type download messages (it expects the next message is sequence (not a resend of a previous msg); this would happen if the NIC was re-sending msgs in a broadcast mode to a failed CMI)
char Improved_DL_Code(void)
{
    unsigned int i;
    unsigned int j,t;
    unsigned char continue_download;
    unsigned char ret_stat;

    continue_download = TRUE;

    switch ( msg_in.number ) /* which message are we processing ? */
    { /*
    case CM1_DL_FILL_PATT: /* fill pattern message */
    } /*
    for ( i=0; i<msg_in.dat.cdl_fill_recv.num_bytes; i++)
        contiguous_dl_data[i] = msg_in.dat.cdl_fill_recv.pattern; /* fill in array */
    Enable_EPROM(); /* enable EProm for writes */
    ret_stat = Page_Write_EPROM((xdata unsigned int*)&msg_in.dat.cdl_fill_recv.addr,
        /* pointer to dest. addr */
        msg_in.dat.cdl_fill_recv.num_bytes, /* DF #4 = num_bytes */
        contiguous_dl_data, /* pointer to source */
        verify_page_write); /* verify flag */
    Protect_EPROM(); /* protect the EE from writes */
    if (ret_stat != SUCCESS)
    { /* the page write failed - indicate failure to the HECU */
        Set_Up_A_Nack( PAGE_WRITE_FAILURE, /* reason */
            ret_stat); /* support data: the failure code */
    } break;
    /*
    case CM1_DL_NORM: /* download data message */
    } /*
    Enable_EPROM(); /* enable EProm for writes */
    ret_stat = 
    Page_Write_EPROM((xdata unsigned int*)&msg_in.dat.cdl_norm_recv.addr,
        /* pointer to dest. addr */
        msg_in.dat.cdl_norm_recv.num_bytes, /* DF #4 = num_bytes */
        &msg_in.dat.cdl_norm_recv.dat[0], /* pointer to source */
        verify_page_write); /* verify flag */
    Protect_EPROM(); /* protect the EE from writes */
    if (ret_stat != SUCCESS)
    { /* the page write failed - indicate failure to the HECU */
        Set_Up_A_Nack( PAGE_WRITE_FAILURE, /* reason */
            ret_stat); /* support data: the failure code */
    } break;
    /*
    case CM1_DL_START_CONT: /* start of a continuation load */
    } /*
    This message allows for the HIC/HECU to send many packets of data, with
    the address and length overhead in just this message. It will be followed by
WM0072139

CM1_CDLCM.CONT msgs until the length is satisfied. Once all data is in, the whole array will be written to EEPROM in a page write.

improved_dl_struct.data_index = MAX_NORM_DL_DATA; /* next place to store */
improved_dl_struct.num_bytes = msg_in.dat.cdl_strt_cont_recv.num_bytes; /* DF # 4 =

num_bytes */
improved_dl_struct.address = (unsigned int)msg_in.dat.cdl_strt_cont_recv.addr;
j=0; /* index into message data */
for ( i=0; i< MAX_NORM_DL_DATA; i++)
{  
    contiguous_dl_data[i] = msg_in.dat.cdl_strt_cont_recv.dat[j++];
}
improved_dl_struct.in_use = TRUE; /* other places check this */
break;

/*-------------------------------------------------------------*/
/* case CM1_CDLCM.CONT: /* a continuation load: see note on CM1_CDLCM.START_CONT */
/*-------------------------------------------------------------*/
i = improved_dl_struct.data_index; /* current index into array */
/* determine how much to read in */
z = ((improved_dl_struct.num_bytes - improved_dl_struct.data_index) < MAX_CONT_DL_DATA)
? (improved_dl_struct.num_bytes - improved_dl_struct.data_index): MAX_CONT_DL_DATA;
for (++j; j< z; j++,++i)
{  
    contiguous_dl_data[i] = msg_in.dat.cdl_strt_cont_recv.dat[j];
    improved_dl_struct.data_index = i; /* next place to store */
}
/* do we have all the data? */
if (i == improved_dl_struct.num_bytes) /* yes: not checking for exact count: ok if its over, we will only */
{ /* write the exact amount to EE anyway */
    Enable_EEPROM(); /* enable EEPROM for writes */
    ret_stat = Page_Write_EEPROM(&improved_dl_struct.address, /* pointer to dest. addr */
     improved_dl_struct.num_bytes,
     contiguous_dl_data, /* pointer to source */
     verify_page_write); /* verify flag */
    Protect_EEPROM(); /* protect the EE from writes */
    /* Clean up the data structure to indicate end */
    improved_dl_struct.in_use = FALSE; /* other places check this */
    if (ret_stat != SUCCESS)
    { /* the page write failed - indicate failure to the HECU */
        Set_Up_A_Nack( PAGE_WRITE_FAILURE, /* reason */
        ret_stat); /* support data: the failure code */
        break;
    }
}
}
break;

/*-------------------------------------------------------------*/
/*-------------------------------------------------------------*/
/* IF the EEPROM has a bad checksum, the ROM code will never jump into EE. This */
/*-------------------------------------------------------------*/
/* message will "Allow" us to jump to EE. IT will jump to code in the last */
/*-------------------------------------------------------------*/
/* EEPROM sector to the Future_Stub() which does a bunch of junk and then returns. */
/*-------------------------------------------------------------*/
/* This area of EE can be used to poke code, diagnostics or whatever, to get the */
/*-------------------------------------------------------------*/
/* ROM to jump to it. What ever happens there is NOT defined here. */
/*-------------------------------------------------------------*/
/* NOTE: Remember that there are 10 data fields in the msg that can be used !!!! */
/*-------------------------------------------------------------*/

Future_Stub();
break;

/*-------------------------------------------------------------*/
/* default: */
/*-------------------------------------------------------------*/
Set_Up_A_Nack( MSG_NOT_SUPPORTED_BY_ROM_CODE, /* reason */
msg_in.number); /* support data: the msg received */
break;

/* end switch */
return(continue_download);
*/
/* end ROM:Download_Code */

13
TITLE: Dnload_Code

DESCRIPTION: This procedure coordinates the downloading of code to EEPROM.
Once this procedure is called, it becomes the main loop. It waits for a download sequence to end cleanly.

INPUTS: load_flag - indicates who the originator is: PROM or EEPROM

OUTPUTS: None

ASSUMPTIONS/LIMITATIONS:

Revision History:
PCSC-122 2/17/98 This procedure now checks to see if Poll_Neuron is returning an error status. If it does, the neuron will be reset.

PCSC-122 2/23/98 Removed the xdata declaration on local variable i
PCSC-174 6/2/98 Removed the timer stuff at the beg of this procedure.
PCSC-213 7/17/98 Removed the clearing of serial communications variables.

End Flush msg type to this procedure

void Dnload_Code(char load_flag)
{
    unsigned char dl_stat;
    dl_stat = TRUE; /* indicate that a download is in progress */

    if (load_flag == RELOAD) /* is entry from eeprom resident code ? */
    {
        /* yes - downloads can start from a request made to the code in EEPROM */
        /* this message was supposed to be processed (accepting broadcast messages) */
        /* for example */
        FeedWd(); /* do not let WDT expire */
    }

    switch( msg_in.number ) /* which message was received ? */
    {
        /* improved download commands (post Version 4 EEPROM code) */
        case CMI_CDL_FILL_PATT: /* indicates a fill pattern */
            dl_stat = Improved_DL_Code(); /* process the download msg */
            break;

        case CMI_CDL_RESET: /* CMl reset instructions */
            dl_stat = Proc_Mess(); /* process the Reset message and acknowledge */
            break;

        case CMI_CDL_FLUSH: /* indicates a "flush" of the EEPROM */
            dl_stat = Proc_Mess(); /* process the Flush message and acknowledge */
            break;

        default: /* these messages are not processed - send back a NACK */
            Send_Bck_Nack( /* Nack the message */
                MSG_NOT_SUPPRTD_BY_ROM_CODE, /* reason code */
                msg_in.number); /* support data */
            break;
    }
    /* end switch */
    /* end if this is a RELOAD */
/* while (dl_stat) *\ continue processing msgs while the download is not complete */
{
    FeedWD(); /* don't let timer expire */
    if (serial_data_flag == TRUE) /* is there an RS232 message avail? */
    {
        /* yes - there is an RS232 message ready */
        memcpy(&msg_in.number,
               &rs232_msg[0],
               sizeof(msg_in));
        dl_stat = Proc_Mess(); /* process the message and acknowledge */
    }
    else /* there is no msg from RS232 avail */
    {
        if (neuron_msg_flag == TRUE) /* is there an HIC/HECU msg avail? */
        {
            /* yes - there is a NEURON message from the HIC/HECU available */
            Flash_the_COMMs_LED: turn it on (on receive); turn off (on response)
            led_save = (led_save & ~COMMs_LED); /* LED on is a clear of just that bit */
            /*(unsigned char xdata *)LED_ADR = led_save; /* updates the actual LEDs */
            Reset_Neuron_Timer();
            Read_Neuron((char xdata *)&msg_in.number);
            Reset_Comms_Timer_O());
            /* prior to processing, check the debug flag to see if we are to dump
               the input message out the RS232 spicket */
            if (dump_neuron_input)
            {
                serial_data_flag = TRUE; /* will be cleared after the send */
                Return_Message(); /* dump it out 232 */
                send_msg_flag = TRUE; /* fix this back */
                neuron_msg_flag = TRUE; /* fix this back */
            }
            dl_stat = Proc_Mess(); /* process the message and acknowledge */
        }
        else /* there is no Neuron message available */
        {
            if (neuron_msg_flag == TIMEOUT_ERROR) /* does Neuron indicate an error? */
            {
                /* yes - reset the Neuron */
                Reset_Neuron();
            }
        }
        /* end if */
        neuron_msg_flag = Poll_Neuron(); /* poll the Neuron for new messages */
    }
}/* endwhile */

/* The download has completed. */

/* This procedure will force the processor to reset, so that on reset, */
/* the prom code will go out and check the EProm code (checksum) matches what */
/* is in the EProm, and if it is valid, it will then jump to it */
/* (unsigned char xdata *)DisEmp0()); /* do not allow the Interrupt handler */
/* to get us out of the infinite loop */
/*(unsigned char xdata *)WFED1 = 0x05; /* reload with a small value */
/* while (1); /* forces a processor reset */

}/* end ROM:Download_Code */

/* END OF CMI BROADCAST DOWNLOAD CODE */

/* BEGIN HECU BROADCAST DOWNLOAD CODE */

*******************************************************************************

15
This class will handle the process of Broadcast Download to all the CMIs on a given NIC. The basic flow is such that the Initiate_Download method is called with the list of desired CMIs and the file to download passed to this method. Error checking is then performed in this method as well as in Perform_File_Validation. Once validation is complete, a series of steps are performed with the use of Perform_Next_Step. Each step must be completed on ALL CMIs before proceeding to the next step. Upon startup, completion, or a failure that causes the process to be aborted, the Initialize_Class method will be called to perform cleanup of the class to prepare it for next use.

This material is considered and contains PROPRIETARY Information of Sanders, a Lockheed-Martin Company. Disclosure to others, use or copying without the express written consent of Sanders is strictly prohibited.

(c) Copyright 1995. Sanders, a Lockheed Martin Company.
Unpublished work. All rights reserved.
/**
 * Name: Fill_Valid_Message_Table
 * Class: Download_CMI_Class
 * Abstract: Function to define the valid messages for each phase
 * Rev History:
 *   | PCR Number | Date | Engineer | Description   |
 *   |-------------|------|----------|---------------|
 *   | PSC-152     | 4/30/98 | Moody   | New           |
 */

void Download_CMI_Class::Fill_Valid_Message_Table()
{
    //initialize them first
    for(int i = 0; i < NUM_MODES; i++)
    {
        for(int j = 0; j < MAX_MSG_TYPES; j++)
        {
            Valid_Message_Types[i][j] = -1;
        }
    }

    //fill in valid values
    for(int k = 0; k < NUM_MODES; k++)
    {
        switch (k)
        {
            case FIND_CMI:
                Valid_Message_Types[k][0] = rua_lost_cmi;
                break;
            case WAKE_UP_CMI:
                Valid_Message_Types[k][0] = act_cmi_msg;
                break;
            case DL_SECTOR_XSUMS:
                case DOWNLOAD_MESSAGES:
                    Valid_Message_Types[k][0] = cmi_cdl_cont;
                    Valid_Message_Types[k][1] = cmi_cdl_fill_pact;
                    Valid_Message_Types[k][2] = cmi_cdl_norm;
                    Valid_Message_Types[k][3] = cmi_cdl_start_cont;
                    Valid_Message_Types[k][4] = cmi_cdl_flush;
                    break;
            case TURN_FLAGS_ON:
                Valid_Message_Types[k][0] = cmi_cdl_stat_instr;
                break;
            case CALC_SECTOR_XSUMS:
                Valid_Message_Types[k][0] = cmi_cdl_calc_sect_xsum;
                break;
            case GET_SECTOR_XSUMS:
                Valid_Message_Types[k][0] = cmi_cdl_rpt_sect_xsum;
                break;
            case VALIDATE_CMI_DL:
                Valid_Message_Types[k][0] = cmi_power_stat_msg;
                break;
            case RESET_CMI:
                Valid_Message_Types[k][0] = cmi_cdl_reset;
                break;
            default:
                case NONE:
                    case FINISHED:
                        case ABORT_MODE:
                            break;
        }
    }
}
void Download_CMI_Class::Initialize_Class()
{
  broadcast_mode = NONE;
  HIC_id = NULL;
  for (int i = 0; i < MAX_CMIS; i++)
  {
    for(int j = 0; j < NUM_SECTORS; j++)
    {
      cmi_number_array[j][i] = DO_NOT_DOWNLOAD;
      cmi_status_array[j][i] = NO_ERROR;
    }
  }
  cmi_counter = 0;
  dl_file[0] = '\0';
  num_cmis = 0;
  num_bytes = 0;
  file_checksum = 0;
  file_pos = 0;
  step_flag = AUTO;
  for (int j = 0; j < NUM_DL_PAGES; j++)
  {
    pages_to_download[j] = DO_NOT_DOWNLOAD;
  }
  //call this to stop timer if it's on and to set reset flag to OFF
  dl_timer_flag = OFF;
  //set the parent class logging flag to ON
  parent_class->log_flag = TRUE;
  //restore parent poll flag
  if(poll_flag == TRUE)
  {
    parent_class->Polling_Enable = TRUE;
    poll_flag = FALSE;
  }
  force_state = FALSE;
}

/*****************************************************************************
 * Name:   Initiate_Download (broadcast)                              *
 * Class:  Download_CMI_Class                                        *
 * Abstract: Begin download for a sector of CMIs in broadcast          *
 * Rev History:                                                      *
  * PCR Number: Date Engineer Description:
  *      3/31/98  PSCE-152 Moody  New                                  *
*****************************************************************************/
int Download_CMI_Class::Initiate_Download(HIC * HICin, int **cmi_ids, char * file,
                        int sector, int force)
{
  int status = NO_ERROR;
  int CMI_IDS[NUM_SECTORS][MAX_CMIS];
  if (cmi_ids)
  {
    memcpy((void *) CMI_IDS, (void *) cmi_ids, NUM_SECTORS * MAX_CMIS * sizeof(int));
  }
  //make sure that we are in a mode that will allow a download
  if ( (broadcast_mode != NONE) && (broadcast_mode != FINISHED) )
  {
    status = DOWNLOAD_IN_PROGRESS;
  }
  else
  {
    //initialize all of the class attributes
    Initialize_Class();
  }
  //set the class member attributes
  HIC_id = HICin;

  return status;
}
strcpy(dll_file, file);
   force_state = force;

// validate file checksum
if(status == NO_ERROR)
   {
      status = Perform_File_Validation(file);
   }

// get and check the CMIs to download
if(status == NO_ERROR)
   {
      if(cmi_ids != NULL)
         {
            for (int j=0; j<NUM_SECTORS; j++)
               {
                  for(int i = 0; i<MAX_CMIS; i++)
                     {
                           if(CMI_IDs[j][i] == GO_DOWNLOAD)
                              {
                                 cmi_number_array[j][i] = CMI_IDs[j][i];
                                 num_cmis++;
                              }
                           else
                              {
                                 cmi_number_array[j][i] = DO_NOT_DOWNLOAD;
                              }
                     }
            
                  if(num_cmis == 0)
                     {
                        status = NO_CMIS_SELECTED;
                     }
                  else
                     {
                        status = NO_CMIS_SELECTED;
                     }
         }
   }

// begin the download process
if(status == NO_ERROR)
   {
      if(parent_class->Polling_Enable == TRUE)
         {
            poll_flag = TRUE;
            parent_class->Polling_Enable = FALSE;
         }
      status = Perform_Next_Step();
   }

// set the remaining attributes
if(status == NO_ERROR)
   {
      // make call to update the interfaces
      parent_class->Update_DL_Status(broadcast_mode, (int **) cmi_status_array);

      // set HEC flag
      parent_class->System_Mode_Flag = DOWNLOAD_MODE;
   }
else
   {
      // set the data back to initialized values
      Initialize_Class();
   }

return(status);

/***********************************************************************************/

* Name:    Perform_File_Validation
* Class:   Download_CMI Class
* Abstract: Verify that the file chosen has valid checksum
int Download_CHI_Class::Perform_File_Validation(char * file)
{
    int status = NO_ERROR;
    ifstream infile;
    ofstream outfile;
    //get handle to file stream and create buffer for line of code
    infile.open(file, ios::in|ios::binary|ios::nocreate);
    //make sure file exists
    if(!infile)
    {
        status = BAD_FILENAME;
    }
    infile.close();
    if(status == NO_ERROR)
    {
        // If Extension == .cd2 / run through CRC-32
        char Extension[5];
        char * Parse_FileName = strrchr(file, ' '); 
        strcpy(Extension, *Parse_FileName);
        char temp_filename[MAX_FILENAME_SIZE];
        if(strcmp(Extension, *cd2) == 0)
        {
            // Get Checksum from file
            char * First_Line;
            char * File_Check_Sum_Ptr;
            unsigned long File_Check_Sum;
            FILE * fpstr;
            fpstr = fopen(file, "rt");
            fgets(First_Line, LINESIZE, fpstr);
            File_Check_Sum_Ptr = strchr(First_Line,':');
            if(File_Check_Sum_Ptr == NULL)
            {
                delete First_Line;
                fclose(fpstr);
                return(INVALID_FILE_DATA);
            }
            File_Check_Sum_Ptr++;
            File_Check_Sum = strtoul(File_Check_Sum_Ptr, NULL, 10);
            // Create CRC Table in Memory
            unsigned long * CRC_Table;
            unsigned long CRC_For_Table;
            unsigned long CRC_Poly;
            int temp, tempj;
            CRC_Table = new unsigned long[256];
            CRC_Poly = 0xEDB88320L;
            for(tempi = 0; tempi < 256; tempi++)
            {
                CRC_For_Table = tempi;
                for(tempj = 0; tempj > 0; tempj--)
                {
                    if(CRC_For_Table & 1)
                    {
                        CRC_For_Table = (CRC_For_Table >> 1) ^ CRC_Poly;
                    }
                    else
                    {
                        CRC_For_Table >>= 1;
                    }
                }
            }
CRC_Table[tempi] = CRC_For_Table;
}

// Calculate crc

int Line_Buffer_Length;
char * Line_Buffer;

Line_Buffer = new char[LINESIZE];
if(Line_Buffer == NULL)
{
   fclose(fpotr);
   delete CRC_Table;
   delete First_Line;
   return(INVALID_FILE_DATA);
}

unsigned long CRC_Save;
unsigned long CRC_Code;

int Trailing = 0;
CRC_Save = CRC_Code = 0xffffffff;
Line_Buffer_Length = 1;
while(fgets(Line_Buffer, LINESIZE, fpotr) != NULL)
{
   char * Limit;
   char * P;
   Line_Buffer_Length = strlen(Line_Buffer);
   Limit = Line_Buffer + Line_Buffer_Length;
   // Search for NewLine Characters and replace with '10'
   for(P = Line_Buffer; P != Limit; P++)
   {
      if(*P == '\n')
      {
         *P = MYNL;
      }
   }
   // Perform CRC algorithm on 1 line of input data
   for(int i = 0; i < Line_Buffer_Length; i++)
   {
   CRC_Code = CRC_Table[(int) CRC_Code] ^ (Line_Buffer[i]) & 0xFF ^ {
   (CRC_Code) >> 8) & 0x00FFFFFF;
   }if(Line_Buffer_Length != 1)
   {
   CRC_Save = CRC_Code;
   }if(!Trailing & Line_Buffer_Length == 1)
   {
   CRC_Code = CRC_Save;
   }
   fclose(fpotr);
   delete CRC_Table;
   delete Line_Buffer;
   delete First_Line;
   if(CRC_Code != File_Check_Sum)
   {
      return(INVALID_FILE_DATA);
   }
   // DO CRC CHECK HERE
   status = NO_ERROR;
   }
Once CRC Check done, need to strip off first line of code from file
// (remember, this is a temp file. The 'real' file is still in the hcdl directory)
strcpy(temp_filename, "C:\hec\temp\temp.bak");
remove(temp_filename);
infile.open(temp_filename, ios::in|ios::binary|ios::nocreate);
outfile.open(temp_filename, ios::out|ios::binary);

//read in first line of infile
char buf[80];
infile.getline(buf,79, '\n');

// Loop through input file and transfer remaining bytes to output file
char ch;
while(infile.get(ch))
{
    if(infile.fail())
    {
        status = INVALID_FILE_DATA;
    }
    outfile.put(ch);
}

// Close the I/O files
infile.close();
outfile.close();

//rename temp file to output file
remove(file);
rename(temp_filename,file);
}

make sure data is valid - check all characters then make sure segment IDs are valid
char buffer[MAX_LINE_SIZE];
char temp_sect[3];
char temp_num[3];
int i;
if(status == NO_ERROR)
{
    infile.open(file, ios::in|ios::binary|ios::nocreate);
    infile.seekg(0);
    while(!infile.eof())
    {
        //get line
        infile.getline(buffer, MAX_LINE_SIZE);
        buffer[CODE_LINE_SIZE] = '\0';

        //validate line size
        if(strlen(buffer) < CODE_LINE_SIZE)
        {
            //make sure that we are not dealing with a last line that is just padded
            //with blanks and newlines
            for(i = 0; i < CODE_LINE_SIZE; i++)
            {
                //see if any valid chars - which there shouldn't be
                if(isxdigit(buffer[i]))
                {
                    status = INVALID_FILE_DATA;
                    break; // out of for loop
                }
                else if(buffer[i] == '\0')
                {
                    break; // out of for loop
                }
            }
        }

        //now see if there is actually another line to be read
        if(!infile.eof())
        {
            status = INVALID_FILE_DATA;
        }
    }
}

// regardless of scenario, break out of while
break; // out of while
//check for valid chars - which they all should be
for(i = 0; i < CODE_LINE_SIZE; i++)
{
    if(!isxdigit(buffer[i]))
    {
        status = INVALID_FILE_DATA;
        break; //out of for loop
    }
}

if(status != NO_ERROR)
{
    break; //out of while
}

//check the message number
temp_num[0] = buffer[0];
temp_num[1] = buffer[1];
temp_num[2] = '\0';

//convert the message number from a hex (radix16) val to a decimal and compare
//to valid CMI DL range
if(((int)strtol(temp_num,NULL,16)) < fut_enhc_d1_msg ||
    ((int)strtol(temp_num,NULL,16)) > cmi_cdl_reset) )
{
    status = INVALID_FILE_DATA;
    break; //out of while
}

//check the sector
temp_sect[0] = buffer[2];
temp_sect[1] = buffer[3];
temp_sect[2] = '\0';

//convert the sector from a hex (radix16) val to a decimal and compare
if(((int)strtol(temp_sect,NULL,16)) > CHECKSUM_SECTOR)
{
    status = INVALID_FILE_SEGMENT;
    break; //out of while
}

infil.close();

return(status);

/*****************************/
// TODO: Add function comments
/*****************************/

50  * Name: Message_Returned
50  * Class: Download_CMI_Class
50  * Abstract: Function indicating that a DL message has returned from HIC/CMI
50  * Rev History:
50  * PCSC-152 3/31/98 Moody New
55  int Download_CMI_Class::Message_Returned(unsigned int data[], int failed_flag)
50  {
50      //retry counters
50      static int reset_retries = 0;
50      static int calc_retries = 0;
50
50      //flag for use after the switch statement
50      int individual_flag = FALSE;
50
50      //status flag for function return
50      int status = NO_ERROR;
50
50      //Make a call to a function that will process the returned message -
50      //pass it the message and the failed flag
50      status = Handle_Return_Data(data, failed_flag);
50
50      //a switch that is based on the mode that we are in at the time - only want to
50      //process if Handle_Return_Data didn't return an error indicating that the message
50      if(status != INVALID_MESSAGE_TYPE)
switch(broadcast_mode)
{
    case (FIND_CMI):
    case (TURN_FLAGS_ON):
    case (GET_SECTOR_XSUMS):
    case (WAKE_UP_CMI):    
    case (VALIDATE_CMI_DL):
        reset_retries = 0;
        calc_retries = 0;
        //set flag for use after switch statement
        individual_flag = TRUE;
        break;

    case (CALC_SECTOR_XSUMS):
        if(status == BROADCAST_MSG_FAILED)
        {
            //see how many in a row
            calc_retries++;
            if(calc_retries == 3)
            {
                calc_retries = 0;
                status = Perform_Next_Step();
                break;
            }
            //retry message
            status = Calc_Sector_Checksums();
        }
        else
        {
            calc_retries = 0;
            status = Perform_Next_Step();
        }
        break;

    case (RESET_CMI):
        if(status == BROADCAST_MSG_FAILED)
        {
            //see how many in a row
            reset_retries++;
            if(reset_retries == 3)
            {
                reset_retries = 0;
                status = Perform_Next_Step();
                break;
            }
            //retry message
            status = Perform_Resets();
        }
        else
        {
            reset_retries = 0;
            status = Perform_Next_Step();
        }
        break;

    case (DL_SECTOR_XSUMS):
    case (DOWNLOAD_MESSAGES):
    case (FINISHED):
        reset_retries = 0;
        calc_retries = 0;
        status = Perform_Next_Step();
        break;

    case (NONE):
    default:
        break;
}

//if the message returned was not a broadcast message (sent to an
//individual CMI), then we need to check some other stuff
if(individual_flag == TRUE)
{
    //see which CMI is responding
    int temp_sector = (data[DIN1]>>6) & 0x03;
}
int cmi_number = data[DIN1] & 0x3F;

// verify that this CMI is supposed to be downloaded
if(cmi_number_array[temp_sector][cmi_number] == GO_DOWNLOAD)

  // if we have found no errors above, then indicate a successful status
  // otherwise set the status flag to whatever the error was
  if(status == NO_ERROR)
    {
      if(cmi_status_array[temp_sector][cmi_number] == OPEN_STATUS)
      {
        cmi_status_array[temp_sector][cmi_number] = GOOD_STATUS;
      }
      else if((cmi_status_array[temp_sector][cmi_number] == CMI_DID_NOT_RESPOND)
      & (broadcast_mode == VALIDATE_CMI_DL))
      {
        cmi_status_array[temp_sector][cmi_number] = GOOD_STATUS;
      }
    }
    else
    {
      cmi_status_array[temp_sector][cmi_number] = status;
    }

  // turn the download flag off for this CMI only if we've failed to
  // find the CMI through the RUA_LOST_CMI message - decrement the
  // number of CMIs we wish to Download
  if((broadcast_mode == FIND_CMIS) && (status != NO_ERROR))
    {
      cmi_number_array[temp_sector][cmi_number] = DO_NOT_DOWNLOAD;
      num_cmis--;
    }
  else
    {
      // increment the counter
      cmi_counter++;
    }

  // see if we're ready to go to next step
  if(cmi_counter == num_cmis)
    {
      // check to see if there are any CMIs to be downloaded (did they all
      // not talk during FIND_CMIS stage??)
      if(broadcast_mode == FIND_CMIS)
        {
          status = COULDN'T_FIND_CMI;
          for(int j = 0; j < NUM_SECTORS; j++)
            {
              for(int i = 0; i < MAX_CMI; i++)
                {
                  if(cmi_number_array[j][i] == GO_DOWNLOAD)
                    {
                      status = NO_ERROR;
                      break;
                    }
                }
            }
      }
      if(status == NO_ERROR)
        {
          // go do the next step in the download process
          status = Perform_Next_Step();
        }
      else
        {
          Cancel_Download();
          return (status);
        }
    }
  else
    {
      status = Perform_Next_Step();
    }

  // make a call to the main window to indicate DL status

if (broadcast_mode != FINISHED) && (broadcast_mode != NONE) )
{
    parent_class->Update_DL_Status(broadcast_mode, (int **) cmi_status_array);
}
return (status);
}

/******************************************************************************
 * Name: Handle_Return_Data
 * Class: Download_CMI_Class
 * Abstract: validate and process the return data from the CMIs
 * Rev History:
 *    PCR Number    Date     Engineer  Description
 *    PCSC-152      4/10/98   Moody    New
*******************************************************************************/
int Download_CMI_CLASS::Handle_Return_Data(unsigned int data[], int failed_flag)
{
    int status = NO_ERROR;

    //get CMI
    int sect = (data[DIN1]>>6) & 0x03;
    int cminum = data[DIN1] & 0x3F;
    CMI * temp_cmi = HIC_id->Child_CMIs[sect][cminum];
    if(temp_cmi == NULL)
    {
      status = NULL_CMI;
    }

    //call function to make sure the message type(s) are valid for the mode we are in
    if (status == NO_ERROR)
    {
      status = Compare_Message_Type(data[MESSAGE_ID], failed_flag);
    }

    //do any other validation of the data that needs to be done
    if (status == NO_ERROR)
    {
      if (failed_flag == FAILED) //want to set status flag to an error condition
      {
        switch (broadcast_mode)
        {
        case FIND_CMI:
          status = COULDN'T_FIND_CMI;
          break;

        case TURN_FLAGS_ON:
          case GET_SECTOR_XSUMS:
          status = CMI_DID_NOT_RESPOND;
          break;

        case WAKE_UP_CMI:
        case VALIDATE_CMI_DL:
          status = INVALID_RETURN_DATA;
          break;

        default:
          case DL_SECTOR_XSUMS:
          case CALC_SECTOR_XSUMS:
          case DOWNLOAD_MESSAGES:
          case RESET_CMI:
            status = BROADCAST_MSG_FAILED;
            break;
        }
      }
    }
    else //not a failed situation - want to perform actions based on message type
    {
      switch(broadcast_mode)
      {
      case TURN_FLAGS_ON:
        //make sure message has flags set properly
        if( (data[DIN1] != ACCEPT_BROADCAST_MSGS) ||
            (data[DIN1] == ACCEPT_DL_MSGS) ||
            (data[DIN7] != CLEAR_REPORT_AREA) )
        {
status = INVALID_RETURN_DATA;
}
break;

case GET_SECTOR_XSUMS:
//store the sectors for which there are bad checksums to indicate which
//sectors to download
    status = Store_Download_Sectors(data);
    break;

case DL_SECTOR_XSUMS:
case DOWNLOAD_MESSAGES:
    break;
case CALC_SECTOR_XSUMS:
case RESET_CMIS:
case FIND_CMIS:
case WAKE_UP_CMIS:
case VALIDATE_CMIS_DL:
default:
    break;
}

return(status);

 /***********************************************************************/
/* Name: Compare_Message_Type */
/* Class: Download_CMIS_Class */
/* Abstract: verify that the message type returned is valid for that phase */
/* Rev History: */
/* PCSC-152 4/30/98 Moody New */
 /***********************************************************************/

int Download_CMIS_Class::Compare_Message_Type(int data_value, int failed_flag)
{
    int status = NO_ERROR;

    if(failed_flag != FAILED)
    {
        int success_flag = 0;

        for (int i = 0; i < MAX_MSG_TYPES; i++)
        {
            if (data_value == Valid_Message_Types[broadcast_mode][i])
            {
                success_flag = 1;
            }
        }

        if(!success_flag)
        {
            status = INVALID_MESSAGE_TYPE;
        }

    }

    return status;
}

 /***********************************************************************/
/* Name: Store_Download_Sectors */
/* Class: Download_CMIS_Class */
/* Abstract: store the sectors of the CHI that need to be downloaded */
/* Rev History: */
/* PCSC-152 5/2/98 Moody New */
 /***********************************************************************/

int Download_CMIS_Class::Store_Download_Sectors(unsigned int data[])
{
    int status = NO_ERROR;

    //make sure that the message checksum matches the DF10 checksum
    unsigned char checksum = 0;
    for(int i = DIN2; i < DIN10; i++)
    {

checksum += data[i];
}
if(checksum != data[DN10]) {
    status = INVALID_MESSAGE_CHECKSUM;
}
if(status == NO_ERROR) {
    // if data[DN9] is VALID (0), then the checksum on sector-57 (which contains
    // the checksums of the other 56 sectors) is OK and we can trust the info in
    // the reset of the bitmap
    if(data[DN9] == 0) {
        for(int i = DIN2; i < DIN9; i++) {
            for(int j = 0; j < 8; j++) {
                if((data[i]>>j)&1) {
                    pages_to_download[8*(i-DIN2)+j] = GO_DOWNLOAD;
                }
            }
        }
    } else //sector that contains the checksums is bad
    {
        status = INVALID_SECTOR_CHECKSUM;
    }
}
return(status);
}"*/

/*****************************************************************************
 * Name:   Perform_Next_Step
 * Class:  Download_CMI_Class
 * Abstract: do the next step in the download process
 * Rev History:
 *   PCR       Date       Engineer   Description
 *   152       4/29/98    Moody      New
*****************************************************************************/
int
Download_CMI_Class::Perform_Next_Step()
{
    int status = NO_ERROR;

    if(step_flag != MANUAL_WAIT) // it must be in AUTO or MANUAL_GO
    {
        switch(broadcast_mode) {
        case NONE:
            status = Find_CMIs();
            break;
        case FIND_CMIS:
            status = Set_CMI_Flags();
            break;
        case TURN_FLAGS_ON:
            // set the parent class logging flag to FALSE
            parent_class->log_flag = FALSE;
            status = DL_Sector_Checksums();
            break;
        case DL_SECTOR_XSUMS:
            // continue to download until an EOF is returned
            status = DL_Sector_Checksums();
            if (status == END_OF_FILE)
            {
                // set the parent class logging flag to ON
                parent_class->log_flag = TRUE;
                // Download is complete, so turn off the broadcast and download flags
                status = Calc_Sector_Checksums();
            }
        }
break;

case CALC_SECTOR_XSUMS:
    //continue to download until an EOF is returned
    if(dl_timer_flag == OFF)
        {  
            //start the DL timer
            status = Set_DL_Timer()
        }
    else
        {  
            //turn off the reset timer flag
            dl_timer_flag = OFF;

            //this was a broadcast message - change all open CMIs back to good
            for(int j=0; j<NUM_SECTORS; j++)
                {  
                    for(int i = 0; i < MAX_CMI; i++)
                        {  
                            if(cmi_status_array[j][i] == OPEN_STATUS)
                                cmi_status_array[j][i] = GOOD_STATUS;
                        }
            }

            //perform next step
            status = Get_Sector_Checksums();
        }
    break;

case GET_SECTOR_XSUMS:
    //set the parent class logging flag to FALSE
    parent_class->log_flag = FALSE;
    status = Download_File();

    //in this special case,
    if(status == ALL SECTORS DOWNLOADED)
        {  
            //set the parent class logging flag to ON
            parent_class->log_flag = TRUE;

            //Download is complete, so turn off the broadcast and download flags
            status = Perform_Resets();
        }
    break;

case DOWNLOAD_MESSAGES:
    //continue to download until an EOF or an ALL SECTORS DOWNLOADED is returned
    status = Download_File();

    if (status == END OF FILE)
        {  
            //current sector is complete - download next sector
            status = Download_File();

            if (status == ALL SECTORS DOWNLOADED)
                {  
                    //set the parent class logging flag to ON
                    parent_class->log_flag = TRUE;

                    //Download is complete, so turn off the broadcast and download
                    status = Perform_Resets();
                }
        }
    break;

case RESET_CMI:
    if(dl_timer_flag == OFF)
        {  
            //start the 15s reset timer
            status = Set_DL_Timer();
        }
    else
        {  
            //...
        }

    //...

29
{  // turn off the reset timer flag
    dl_timer_flag = OFF;

    // this was a broadcast message - change all open CMIs back to good
    for (int j=0; j<NUM_SECTORS; j++)
    {
        for (int i = 0; i < MAX_CMI; i++)
        {
            if(cmi_status_array[j][i] == OPEN_STATUS)
            {
                cmi_status_array[j][i] = GOOD_STATUS;
            }
        }
    }

    // perform next step
    status = Wake_Up_CMIs();
    break;
    case WAKE_UP_CMI:
    status = Validate_CMI_DL();
    break;
    case VALIDATE_CMI_DL:
    status = Download_Complete();
    break;
    case FINISHED:
    default:
    break;
        }
    return status;
}

******************************************************************************
* Name: Find_CMIs
* Class: Download_CMI_Class
* Abstract: Send messages to all of the CMIs to find them
* Rev History:
*   PCR Number Date Engineer Description
*   PCSC-152 4/1/98 Moody New
******************************************************************************
int Download_CMI_Class::Find_CMIs()
{
    int status = NO_ERROR;

    // build message to send to CMIs
    MSG_PKT msg_out;
    msg_out.message_id = rua_lost_cmi;
    msg_out.message_tag = 0;
    for (int j = DOUT2; j < MAX_DATA_BYTES; j++)
    {
        msg_out.data[j] = 0;
    }

    // set counter to 0
    cmi_counter = 0;

    // set mode flag
    broadcast_mode = FIND_CMIS;

    for (j = 0; j < NUM_SECTORS; j++)
    {
        msg_out.data[DOUT2] = HIC_id->US_Primary_Channel[j];
        msg_out.data[DOUT3] = HIC_id->US_Diversity_Channel[j];

        // send messages to CMI
        for (int i = 0; i < MAX_CMI; i++)
        {
            if(cmi_number_array[j][i] == GO_DOWNLOAD)
            {
                // set CMI ID in data field
                msg_out.data[DOUT1] = (j<<6) + i;
            }
        }
    }

    return status;
}
// start the CMIs out as all being open
  cmi_status_array[j][i] = OPEN_STATUS;

  // place messages into queue
  parent_class->Send_Message(msg_out, HIC_id, NO_POLL, INTERNAL, -1, -1, TRUE);
}
}

return(status);
}

/******************************************************************************
 * Name:       Set_CMI_Flags
 * Class:      Download_CMI_Class
 * Abstract:   Send messages to all of the CMIs to set their Broadcast flags
 * Rev History:
 *              
 * PCR Number  Date       Engineer    Description
 *              20        3/31/98    Moody      New
*******************************************************************************/

int Download_CMI_Class::Set_CMI_Flags()
{
  int status = NO_ERROR;

  // build message to send to CMIs
  MSG_PKT msg_out;
  msg_out.message_id = cmi_cdl_stat_instr;
  msg_out.message_tag = 0;
  msg_out.data[DOUT2] = 0x0B; // data integrity
  msg_out.data[DOUT3] = ACCEPT_BROADCAST_MSGS;
  msg_out.data[DOUT4] = 0x0E; // data integrity
  msg_out.data[DOUT5] = ACCEPT_DL_MSGS;
  msg_out.data[DOUT6] = 0x0E; // data integrity
  msg_out.data[DOUT7] = CLEAR_REPORT_AREA;
  msg_out.data[DOUT8] = 0x0F; // data integrity
  msg_out.data[DOUT9] = 0;
  msg_out.data[DOUT10] = 0;

  // set counter to 0
  cmi_counter = 0;

  // set mode flag
  broadcast_mode = TURN_FLAGS_ON;

  for (int j = 0; j < NUM_SECTORS; j++)
  {
    // send messages to CMI
    for (int i = 0; i < MAX_CMIS; i++)
    {
      if(cmi_number_array[j][i] == GO_DOWNLOAD)
      {
        // set CMI id in data field 1
        msg_out.data[DOUT1] = (j<<6) + i;

        // set status flag to 'open' if it hasn't failed any steps so far
        if(cmi_status_array[j][i] == GOOD_STATUS)
        {
          cmi_status_array[j][i] = OPEN_STATUS;
        }

        // place messages into queue
        parent_class->Send_Message(msg_out, HIC_id, NO_POLL, INTERNAL, -1, -1, TRUE);
      }
    }
  }

  return(status);
}

/******************************************************************************
 * Name:       DL_Sector_Checksums
 * Class:      Download_CMI_Class
 * Abstract:   Send messages to the CMIs to download the sector checksums
 * Rev History:
 *              
 * PCR Number  Date       Engineer    Description
 *              75        3/31/98    Moody      New
*******************************************************************************/
int Download_CMI_Class::DL_Sector_Checksums()
{
    int status = NO_ERROR;
    int i, j;

    //set broadcast mode
    broadcast_mode = DL_SECTOR_XSUMS;

    //this function handles the downloading of a given sector. Need to scan the
    //download file to find those lines
    status = Download_Sector(CHECKSUM_SECTOR);

    //see if status returned indicates END_OF_FILE or an error
    switch (status)
    {
    case COULDNT_FIND_FILE_SECTOR:
    case INVALID_FILE_DATA:
    case INCORRECT_SECTOR_REQUEST:
        //since this is a critical step in the download process, we need
        //to abort if we got an error in the Download_Sector file
        for (j = 0; j < NUM_SECTORS; j++)
        {
            for (i = 0; i < MAX_CMIS; i++)
            {
                if (cmi_status_array[j][i] == OPEN_STATUS)
                {
                    cmi_status_array[j][i] = status;
                }
            }
        }
        break;

    case NO_ERROR:
        //this is a broadcast message - change all good CMIs to open
        for (j = 0; j < NUM_SECTORS; j++)
        {
            for (i = 0; i < MAX_CMIS; i++)
            {
                if (cmi_status_array[j][i] == GOOD_STATUS)
                {
                    cmi_status_array[j][i] = OPEN_STATUS;
                }
            }
        }
        break;

    case END_OF_FILE:
        //this is a broadcast message - change all open CMIs back to good
        for (j = 0; j < NUM_SECTORS; j++)
        {
            for (i = 0; i < MAX_CMIS; i++)
            {
                if (cmi_status_array[j][i] == OPEN_STATUS)
                {
                    cmi_status_array[j][i] = GOOD_STATUS;
                }
            }
        }
        break;

    default:
        //do nothing
        break;
    }

    return (status);
}

******************************************************************************
* Name:          Calc_Sector_Checksums
* Class:         Download_CMI_Class
* Abstract:      Send messages to the CMIs to instruct that they calculate
                 the sector checksums
* Rev History:   
int Download_CMI_Class::Calc_Sector_Checksums()
{
    int status = NO_ERROR;

    // if user chose to force all checksums to be downloaded, then ignore steps
    // or asking CMIs to calculate them and go right to download step
    if(force_state == TRUE)
    {
        for(int j = 0; j < NUM_DL_PAGES; j++)
        {
            pages_to_download[j] = GO_DOWNLOAD;
        }
        Download_File();
        return status;
    }

    // set mode flag and make sure timer flag is OFF
dl_timer_flag = OFF;
broadcast_mode = CALC_SECTOR_XSUMS;

    // build message to send to CMIs
    MSG_PKT msg_out;
    msg_out.message_id = cmi_cdl_calc_sect_xsum;
    msg_out.message_tag = 0;
    msg_out.data[DOUT2] = 0x0B;  // data integrity
    msg_out.data[DOUT3] = 0;
    msg_out.data[DOUT4] = 0x0E;  // data integrity
    msg_out.data[DOUT5] = 0;
    msg_out.data[DOUT6] = 0x0E;  // data integrity
    msg_out.data[DOUT7] = 0;
    msg_out.data[DOUT8] = 0x0F;  // data integrity
    msg_out.data[DOUT9] = 0;
    msg_out.data[DOUT10] = 0;

    // set counter to 0
    cmi_counter = 0;

    // broadcast message
    msg_out.data[DOUT1] = BROADCAST_MASK;
    parent_class->Send_Message(msg_out, HIC_id, NO_POLL, INTERNAL, -1, -1, TRUE);

    // this is a broadcast message - change all good CMIs to open
    for (int j = 0; j < NUM_SECTORS; j++)
    {
        for (int i = 0; i < MAX_CMIS; i++)
        {
            if(cmi_status_array[j][i] == GOOD_STATUS)
            {
                cmi_status_array[j][i] = OPEN_STATUS;
            }
        }
    }

    return (status);
}

******************************************************************************
 * Name:   Get_Sector_Checksums
 * Class:  Download_CMI_Class
 * Abstract: Send messages to the CMIs to get their sector xsum flags
 * Rev History:
 *    PCR Number   Date        Engineer   Description
 *    PCSC-152     4/30/98     Moody     New
******************************************************************************

int Download_CMI_Class::Get_Sector_Checksums()
{
    int status = NO_ERROR;

    // build message to send to CMIs
    MSG_PKT msg_out;
    msg_out.message_id = cmi_cdl_rpt_sect_xsum;
    msg_out.message_tag = 0;
for(int i = DOUT2; i < MAX_DATA_BYTES; i++)
{
    msg_out.data[i] = 0;
}

//set counter to 0
cmi_counter = 0;

//set mode flag
broadcast_mode = GET_SECTOR_XSUMS;

for (int j = 0; j < NUM_SECTORS; j++)
{
//send messages to CMIs
for (int i = 0; i < MAX_CMIS; i++)
{
    if(cmi_number_array[j][i] == GO_DOWNLOAD)
    {
        //set CMI id in datafield 1
        msg_out.data[DOUT1] = (j<<6) + i;

        //set status flag to 'open' if it hasn't failed any steps so far
        if(cmi_status_array[j][i] == GOOD_STATUS)
        {
            cmi_status_array[j][i] = OPEN_STATUS;
        }

        //place messages into queue
        parent_class->Send_Message(msg_out, HIC_id, NO_POLL, INTERNAL,-1,-1,TRUE);
    }
}
}

return (status);

****************************************************************************
* Name:    Download_File
* class:   Download_CMI_Class
* Abstract: Read download file and broadcast messages to the CMIs
* Rev History:
*   PCR Number Date Engineer Description
*   PCSC-152 4/7/98 Moody  New
****************************************************************************
int
Download_CMI_Class::Download_File()
{
    int status = NO_ERROR;
    int i, j;

    //a static variable to keep track of which sector we are downloading
    static int dl_sector = -1;

    //set the dl_sector
    if(dl_sector == -1)
    {
        int i = 0;
        while((pages_to_download[i] == DO_NOT_DOWNLOAD) && (i < NUM_DL_PAGES) )
        {
            i++;
        }

        if(i != NUM_DL_PAGES)
        {
            //want to set the sector_to_dl flag to 'i', which was determined to
            //be the next sector to download. Also want to set that sector's
            //dl_flag to false, so it won't get re-downloaded
            dl_sector = i;
            pages_to_download[i] = DO_NOT_DOWNLOAD;
        }
        else
        {
            //no more sectors to download
            status = ALL_SECTORS_DOWNLOADED;
        }
    }

//set mode
broadcast_mode = DOWNLOAD_MESSAGES;

if (status != NO_ERROR)
{
    //this function handles the downloading of a given sector. Need to scan the
    //download file to find those lines
    status = Download_Sector(dl_sector);
}

//see if status returned indicates END_OF_FILE or an error
switch(status)
{
    case END_OF_FILE:
    case COULDN'T_FIND_FILE_SECTOR:
    case INVALID_FILE_DATA:
    case INCORRECT_SECTOR_REQUEST:
        //if END_OF_FILE, we want to set the dl_sector flag back to -1 so it will
        //determine the next sector to download, if one of the error conditions,
        //want to skip this sector and go to the next sector
        dl_sector = -1;
        break;

    case NO_ERROR:
        //this is a broadcast message - change all good CMIs to open
        for (j = 0; j < NUM_SECTORS; j++)
        {
            for (i = 0; i < MAX_CMI; i++)
            {
                if(cm_status_array[j][i] == GOOD_STATUS)
                {
                    cm_status_array[j][i] = OPEN_STATUS;
                }
            }
        }
        break;

    case ALL_SECTORS_DOWNLOADED:
        //this is a broadcast message - change all open CMIs back to good
        for (j = 0; j < NUM_SECTORS; j++)
        {
            for (i = 0; i < MAX_CMI; i++)
            {
                if(cm_status_array[j][i] == OPEN_STATUS)
                {
                    cm_status_array[j][i] = GOOD_STATUS;
                }
            }
        }
        dl_sector = -1;
        break;

    default:
        //don't do anything - the sector is not finished being downloaded
        break;
}

return(status);
}
msg_out.data[DOUT0] = 0x00;    /*data integrity
msg_out.data[DOUT1] = 0;
msg_out.data[DOUT2] = 0x00;    /*data integrity
msg_out.data[DOUT3] = 0x00;
msg_out.data[DOUT4] = 0x00;
msg_out.data[DOUT5] = 0x00;
msg_out.data[DOUT6] = 0x00;
msg_out.data[DOUT7] = 0x00;
msg_out.data[DOUT8] = 0x00;
msg_out.data[DOUT9] = 0x00;
msg_out.data[DOUT10] = 0;

//set counter to 0
cmi_counter = 0;

//set mode flag
dl_timer_flag = OFF;
broadcast_mode = RESET_CMIS;

//broadcast message
msg_out.data[DOUT1] = BROADCAST_MASK;
parent_class->Send_Message(msg_out, MCI_ID, NO_POLL, INTERNAL, -1, -1, TRUE);

//this is a broadcast message - change all good CMIs to open
for (int j = 0; j < NUM_SECTORS; j++)
{
    for (int i = 0; i < MAX_CMIS; i++)
    {
        if (cmi_status_array[j][i] == GOOD_STATUS)
            cmi_status_array[j][i] = OPEN_STATUS;
    }
}
return (status);
}

/*******************************************************************************
* Name: Wake_Up_CMIS
* Class: Download_CMI_Class
* Abstract: Function to re-activate CMIS
* Rev History:
* PCR Number Date Engineer Description
*******************************************************************************/

int Download_CMI_Class::Wake_Up_CMIS()
{
    int status = NO_ERROR;

    //build message to send to CMIs
    CMI * temp_cmi;
    MSG_PKT msg_out;
    msg_out.message_id = act_cmi_msg;
    msg_out.message_tag = 0;
    for (int j = DOUT0; j < MAX_DATA_BYTES; j++)
    {
        msg_out.data[j] = 0;
    }

    //set counter to 0
    cmi_counter = 0;

    //set mode flag
    broadcast_mode = WAKE_UP_CMIS;

    //build common part of message structure
    msg_out.data[4] = MCI_ID->DS_Channel;
    msg_out.data[5] = MCI_ID->DS_Channel_Type;
    msg_out.data[6] = (MCI_ID->PC5_Channel1792) >> 8;
    msg_out.data[7] = MCI_ID->PC5_Channel1255;
    msg_out.data[8] = 0;
    for (j = 0; j < NUM_SECTORS; j++)
    {
        //build sector portion of message structure
        msg_out.data[2] = MCI_ID->US_PRIMARY_Channel[j];
        msg_out.data[3] = MCI_ID->US_DIVERSITY_Channel[j];
    }
//loop thru CMIs
for (int i = 0; i < MAX_CMIS; i++)
{
    if(cmi_number_array[j][i] == GO_DOWNLOAD)
    {
        temp_cmi = HIC_id->Child_CMIs[j][i];
        if(temp_cmi != NULL)
        {
            //set status flag to 'open' if it hasn't failed any steps so far
            if(cmi_status_array[j][i] == GOOD_STATUS)
            {
                cmi_status_array[j][i] = OPEN_STATUS;
            }

            //build CMI portion of message structure
            msg_out.data[DOUT1] = (j<<6) + i;
            msg_out.data[DOUT2] = temp_cmi->State +
            (temp_cmi->Autostat<<1) +
            (temp_cmi->DS_Autogain<<2) +
            (LAST_KNOWN_STATE<<3) +
            (temp_cmi->Alarm_Disposition<<4) +
            (temp_cmi->US_Autogain<<5);
            msg_out.data[DOUT9] = temp_cmi->PA_State +
            (temp_cmi->Prim_Rcv_State<<1) +
            (temp_cmi->Div_Rcv_State<<2);

            //place messages into queue
            parent_class->Send_Message(msg_out, HIC_id, NO_POLL, INTERNAL,-1,-
            1,TRUE);
            else
            {
                cmi_status_array[j][i] = COULDNT_FIND_CMI;
            }
        }
    }
}

return(status);

.isNullOrEmpty()
{
    //send messages to CMI
    for (int i = 0; i < MAX_CMIS; i++)
    {
        if(cmi_number_array[j][i] == GO_DOWNLOAD)
        {
            //set CMI id in datafield 1.
            msg_out.data[DOFT1] = (j<<6) + i;

            //set status flag to 'open' if it hasn't failed any steps so far
            if(cmi_status_array[j][i] == GOOD_STATUS)
            {
                cmi_status_array[j][i] = OPEN_STATUS;
                cmi_status_array[j][i] = OPEN_STATUS;
            }

            //place messages into queue
            parent_class->Send_Message(msg_out, HIC_id, NO_POLL, INTERNAL, -1, -1, TRUE);
        }
    }
}
return (status);
}

 /**************************************************************************
 * Name:         Download_Complete
 * Class:        Download_CMI Class
 * Abstract:     Function to send status requests to CMI
 * Rev History:  
 * PCSC-152      4/10/98     Moody     New
 *> **************************************************************************/

 int Download_CMI::Download_Complete(int cancel_flag)
 {
    int status = NO_ERROR;

    MSG_PKT msg_out;
    msg_out.message_tag = 0;
    for (int j = 0; j < MAX_DATA_BYTES; j++)
    {
        msg_out.data[j] = 0;
    }

    if(cancel_flag != NOT_CANCEL)
    {
        //set mode flag
        broadcast_mode = FINISHED;

        //need to finish by sending status messages to each of the CMIs
        //to get the HECU updated
        for (int j = 0; j < NUM SECTORS; j++)
        {
            //now send out status update messages
            msg_out.data[DOFT2] = HIC_id->US_PRIMARY.Channel[j];

            for (int i = 0; i < MAX_CMIS; i++)
            {
                if(cmi_number_array[j][i] == GO DOWNLOAD)
                {
                    msg_out.data[DOFT1] = (j<<6) + i;

                    //send attenuator status message
                    msg_out.message_id = cmi_att_stat msg;
                    parent_class->Send_Message(msg_out, HIC_id, NO_POLL, INTERNAL, -1, -1, TRUE);

                    //send cmi version number message
                    msg_out.message_id = cmi_read_ver_msg;
                    parent_class->Send_Message(msg_out, HIC_id, NO_POLL, INTERNAL, -1, -1, TRUE);

                    //send freq stat message
                    msg_out.message_id = cmi_freq_stat_msg;
                    parent_class->Send_Message(msg_out, HIC_id, NO_POLL, INTERNAL, -1, -1, TRUE);

                    //send power status message
                    msg_out.message_id = cmi_power_stat_msg;
                }
            }
        }
    }
}
msg_out.data[DOUT3] = HIC_id->US_Diversity_Channel[j];
parent_class->Send_Message(msg_out, HIC_id, NO_POLL, INTERNAL, -1, -1, TRUE);

// send message to get cmi faults in read mode
msg_out.message_id = cmi_enable_faults;
msg_out.message_tag = 0;
msg_out.data[DOUT2] = 0;
msg_out.data[DOUT3] = 0;
msg_out.data[DOUT10] = 0;
parent_class->Send_Message(msg_out, HIC_id, NO_POLL, INTERNAL, -1, -1, TRUE);

// Send Messages to get cmi fault limits in read mode
msg_out.message_id = cmi_set_fault_limit_2;
msg_out.message_tag = 0;
msg_out.data[DOUT10] = 0;
parent_class->Send_Message(msg_out, HIC_id, NO_POLL, INTERNAL, -1, -1, TRUE);
msg_out.message_id = cmi_set_fault_limit;
parent_class->Send_Message(msg_out, HIC_id, NO_POLL, INTERNAL, -1, -1, TRUE);

else
    {
    // set mode flag
    broadcast_mode = ABORT_MODE;

    //send message out to turn off broadcast/download flags
    msg_out.message_id = cmi_cd thaw instr;
    msg_out.data[DOUT2] = 0x0B; // data integrity
    msg_out.data[DOUT3] = DO NOT ACCEPT BROADCAST MSGS;
    msg_out.data[DOUT4] = 0x06; // data integrity
    msg_out.data[DOUT5] = DO NOT ACCEPT DL MSGS;
    msg_out.data[DOUT6] = 0x0E; // data integrity
    msg_out.data[DOUT7] = DO NOT CLEAR RPT AREA;
    msg_out.data[DOUT8] = 0x0F; // data integrity
    msg_out.data[DOUT9] = 0;
    msg_out.data[DOUT10] = 0;

    //broadcast message
    msg_out.data[DOUT1] = BROADCAST MASK;
    parent_class->Send_Message(msg_out, HIC_id, NO_POLL, INTERNAL, -1, -1, TRUE);

    //need to reset function that handles downloading of code
    Download_Sector(0, TRUE);
    }

//call function to notify parent class (HEC) that the download is complete
parent_class->Update_DL_Status(broadcast_mode, (int **) cmi_status_array);

//turn off mode flag
parent_class->System_Mode_Flag = NO_MODE;

//call function to re-initialize all class members
Initialize_Class();

return (status);
}

******************************************************************************
* Name:     Download_Sector
* Class:    Download_CMI_Class
* Abstract: Function to download a particular sector of the CMI
* Rev History:
*    PCR Number  Date   Engineer         Description
*    102-152     5/6/98    Moody           New
******************************************************************************

int
Download_CMI_Class::Download_Sector(int sector, int cancel_flag)
{
    //status flag to return
    int status = NO_ERROR;

    //temp buffers
    char buffer[MAX_LINE_SIZE];
char temp_sect[3];

// this variable will let us know if we need to go to a different part of
// the file or continue with the portion we are working on
static int current_sector = -1;

// if we cancelled the download, we want to set the "current_sector" flag
// to -1 so that on the next download we won’t get errors.
if(cancel_flag == TRUE)
{
    current_sector = -1;
    return status;
}

// get handle to file stream and create buffer for line of code
ifstream infile([file], ios::in|ios::binary|ios::nocreate);

// if the current sector equals the sector passed in, it is a normal download
// else - if the current sector is NOT equal to the passed sector, then we need
// to check to be sure that current_sector is -1, indicating that there is
// not a DL in progress for a different sector
if(current_sector == sector)
{
    // set the file pointer to the last known position, get the code,
    // store the new position
    infile.seekg(file_pos);
    infile.getline(buffer, MAX_LINE_SIZE);
    buffer[CODE_LINE_SIZE] = '0';
    file_pos = infile.tellg();

    // make sure buffer size is OK
    if (strlen(buffer) != CODE_LINE_SIZE)
    {
        // check code size
        if (strlen(buffer) == 0)
            status = END_OF_FILE;
        else
            status = INVALID_FILE_DATA;

        // whether or not line is good, set sector = -1
        current_sector = -1;
    }

    // see if the sector field is the same as the current sector
    if(status == NO_ERROR)
    {
        temp_sect[0] = buffer[2];
        temp_sect[1] = buffer[3];
        temp_sect[2] = '0';

        if ((int)strtol(temp_sect, NULL, 16) != current_sector)
        {
            // reset the current sector and set status flag to indicate there
            // are no more lines in this sector
            current_sector = -1;
            status = END_OF_FILE;
        }
    }
}
else // current_sector != sector
{
    if(current_sector != -1)
    {
        // current sector was already set to a different sector
        current_sector = -1;
        status = INCORRECT_SECTOR_REQUEST;
    }
 else // legitimately starting a new sector
{
    // reassign the current sector
    current_sector = sector;

    // set the file pointer to the beginning of the file
    file_pos = 0;
    infile.seekg(file_pos);

    // search through file until find a line with correct sector id
    int success_flag = FALSE;
do;
    //store current position, get line, store new position
    infile.getline(buffer, MAX_LINE_SIZE);
    buffer[CODE_LINE_SIZE] = '\0';
    file_pos = infile.tellg();

    //get the sector
    temp_sector[0] = buffer[2];
    temp_sector[1] = buffer[3];
    temp_sector[2] = '\0';

    //convert the sector from a hex (radix16) val to a decimal and compare
    if(((int)strtol(temp_sector,NULL,16)) == current_sector)
    {
      success_flag = TRUE;
      break;
    }
  } while ( strlen(buffer) == (CODE_LINE_SIZE) );

  //see if we did not break out of do/while loop
  if(success_flag == FALSE)
  {
    status = COULDNT_FIND_FILE_SECTOR;
    current_sector = -1;
  }
}

//close the file
infile.close();

//send message to the CMIs in broadcast mode
if(status == NO_ERROR)
{
  int i = 0, j = 0, index = 0;
  char temp_str[3];
  int datafield[DATA_PACKET_SIZE];

  //want to get two characters at a time, stopping at the return char - also
  //use the check of 'i' for CODE_LINE_SIZE as a boundary check
  while ((buffer[i] != '\r') && (i < CODE_LINE_SIZE))
  {
    while (j < 2)
    {
      temp_str[j] = buffer[i];
      j++;
      i++;
    }

    temp_str[j] = '\0';
    j = 0;

    datafield[index] = (unsigned char)strtol(temp_str,NULL,16);
    index++;
  }

  //build the download message
  MSG_PKT msg_out;
  msg_out.message_id = (MSG_ID)datafield[MESSAGE_ID];
  msg_out.message_tag = 0;

  //set DF #2 thru DF #10
  for(i = DOUT2; i < MAX_DATA_BYTES; i++)
  {
    msg_out.data[i] = (unsigned char)datafield[i+OVERHEAD_BYTES];
  }

  //broadcast message
  msg_out.data[DOUT1] = BROADCAST_MASK;
  parent_class->Send_Message(msg_out, HIC_id, NO_POLL, INTERNAL,-1,-1,TRUE);
}

return (status);
}

******************************************************************************
* Name:    Set_Du_Timer
int Download_CMI_Class::Set_DL_Timer()
{
    int status = NO_ERROR;

    //if SetTimer returns a 0, operation has failed, so set status flag accordingly
    if (parent_class->Set_DL_Timer() == 0)
    {
        status = COULDN'T_SET_DL_TIMER;
    }
    else
    {
        dl_timer_flag = ON;
    }

    return (status);
}

/*******************************************************************************/
*	Name: Stop_Reset_Timer
*	Class: Download_CMI_Class
*	Abstract: Function to stop the CMI reset timer
*	Rev History:
*	    PCR Number	Date	Engineer	Description
*	    PCSC-152	5/13/98	Moody	New
*******************************************************************************/
void Download_CMI_Class::Stop_DL_Timer()
{
    //Call to parent class which will call parent window to stop timer
    parent_class->Stop_DL_Timer();
}

/*******************************************************************************/
*	Name: DL_Timer_Complete
*	Class: Download_CMI_Class
*	Abstract: Outside class indicating that reset timer is complete
*	Rev History:
*	    PCR Number	Date	Engineer	Description
*	    PCSC-152	4/29/98	Moody	New
*******************************************************************************/
int Download_CMI_Class::DL_Timer_Complete()
{
    int status = NO_ERROR;

    if (dl_timer_flag == ON)
    {
        //make call to perform next step
        status = Perform_Next_Step();
    }

    if (status != NO_ERROR)
    {
        Cancel_Download();
    }
    else
    {
        parent_class->Update_DL_Status(broadcast_mode, (int **) &cmi_status_array);
    }

    return(status);
}

/*******************************************************************************/
*	Name: Cancel_Download
*	Class: Download_CMI_Class
*	Abstract: Outside class indicating that reset timer is complete
*	Rev History:
*	    PCR Number	Date	Engineer	Description
*	    PCSC-152	4/29/98	Moody	New
*******************************************************************************/
int Download_CMD_Class::Cancel_Download()
{
    Stop_DL_Timer();
    Download_Complete(CANCEL);
    return(NO_ERROR);
}

int Download_CMD_Class::Get_Step_State()
{
    return(step_flag);
}

int Download_CMD_Class::Set_Step_State(int flag)
{
    step_flag = flag;
    return(NO_ERROR);
}

It will be appreciated that the above described system is more efficient in the downloading of information to the cable microcell integrators due to the hybrid unicast/broadcast mode of operation. Additional efficiencies in the downloading of information include the data formatting techniques indicated in the program listing below, written in C and batch instructions. Of importance is one efficiency due to the fact that in the broadcast mode addresses are not necessary. This frees up the address blocks so that data can be placed therein.

By way of overview, the purpose of the code referred to as HEX2CDL is to convert the output file of a cable microcell integrator compilation (HEX) to a CDL file format that takes advantage of the cable microcell integrator's broadcast download messages. CDL messages are ordered in the CDL file in such a way as to allow the head end control unit to quickly download checksums of a new image to the cable microcell integrators and to allow easy access and sending
of any specific sector of the cable microcell integrator code. Each sector of this code is represented by the smallest number of CDL messages possible.

In detail, what follows is a list of the tasks that are done by the HEX2CDL code. First, the HEX2CDL code separates the ROM code from the EEPROM code. Note that ROM code does not get downloaded. The EEPROM is broken into Fifty-Seven 512 byte sectors (0-56 inclusive). A 32 bit checksum is calculated for each sector. A checksum is placed inside messages marked as sector 56 (0x38). These checksum messages are in order, Sector 0 first, and are placed at the top of the CDL file. Each Sector of EEPROM code is processed to produce the minimum number of CDL messages and Run Length Encoding. All messages pertaining to a Sector are grouped together. The tag field of each message contains the sector number, which is how the head end control unit can download any specific sector. Each line of the CDL is a separate message all ready to be sent by the head end control unit.

More particularly, the following is a sample CDL Output file. Comments are to the right of // and are not part of the file. Typically there are about 3000 lines to a CDL file. The 3rd and 4th Character in each line is considered to be the TAG byte, 0x38 = sector 56 which tells the HECU it is checksum information. The F000 is the address to write

```
3B3800F000E400010CD50001  //Checksum information in each line until the tag = 0
373800033700000D2AF0000E9  //each checksum is 4 bytes,
37380029000107FD0000F8AD
.
37380000001FE000001FE0000
37380001FE000001FE000001
373800F0A000028C3000000
3B00008000F9E4906771F0E4  //first line of Sector 0, write it at address 0x8000
37000090676FF0E4906770F0  //The above message is a Start of block 0x38, this message is 0x37 Block
.
393500EA00FCFF0000000000  // There is no code in sector 0x35 (53 decimal) FF is the blank Character
```

44
393500E4FCF0000000000 // One wants to erase anything that might have been
there before
393500EB806F0000000000 // One can accomplish replacing 512 bytes with these
3A3500EBF8000000000 // 4 messages
393600EC00F0000000000 // Start of Sector 54, it too is blank
393600EFCF0000000000
393600EDF806F0000000000
3A3600EDF802FFF0000000000
393700EE00FCF0000000000
393700EF2680F0000000000
3B3700E2F2AFFF0000000000 // Some code exists in sector 55 starting at
address 0xEF
373700DA000F9066DC
373700F007C00EB2DFE3A3C
373700F3066D000A3EFF0
373700FAD02508F50822FFF
3A3700EFFE020F0000000000

20 rem
rem BLOCKSEG.BAT
rem
rem The batchfile makes 55 calls of the blockout executable to process
rem all the files created by condseg.bat
25 rem
rem
blockout seg0000.zzc seg0000.blo
blockout seg0001.zzc seg0001.blo
blockout seg0002.zzc seg0002.blo
30 blockout seg0003.zzc seg0003.blo
blockout seg0004.zzc seg0004.blo
blockout seg0005.zzc seg0005.blo
blockout seg0006.zzc seg0006.blo
blockout seg0007.zzc seg0007.blo
35 blockout seg0008.zzc seg0008.blo
blockout seg0009.zzc seg0009.blo
blockout seg000A.zzc seg000A.blo
blockout seg000B.zzc seg000B.blo
blockout seg000C.zzc seg000C.blo
40 blockout seg000D.zzc seg000D.blo
blockout seg000E.zzc seg000E.blo
blockout seg000F.zzc seg000F.blo
blockout seg0010.zzc seg0010.blo
45 blockout seg0011.zzc seg0011.blo
blockout seg0012.zzc seg0012.blo
blockout seg0013.zzc seg0013.blo
blockout seg0014.zzc seg0014.blo
blockout seg0015.zzc seg0015.blo
50 blockout seg0016.zzc seg0016.blo
blockout seg0017.zzc seg0017.blo
blockout seg0018.zzc seg0018.blo
blockout seg0019.zzc seg0019.blo
blockout seg001A.zzc seg001A.blo
blockout seg001B.zzc seg001B.blo
55 blockout seg001C.zzc seg001C.blo
blockout seg001D.zzc seg001D.blo
blockout seg001E.zzc seg001E.blo
blockout seg001F.zzc seg001F.blo
blockout seg0020.zzc seg0020.blo
60 blockout seg0021.zzc seg0021.blo
blockout seg0022.zzc seg0022.blo
blockout seg0023.zzc seg0023.blo
blockout seg0024.zzc seg0024.blo
blockout seg0025.zzc seg0025.blo
blockout seg0026.zzc seg0026.blo
blockout seg0027.zzc seg0027.blo
5     blockout seg0028.zzc seg0028.blo
blockout seg0029.zzc seg0029.blo
blockout seg002A.zzc seg002A.blo
blockout seg002B.zzc seg002B.blo
blockout seg002C.zzc seg002C.blo
10    blockout seg002D.zzc seg002D.blo
blockout seg002E.zzc seg002E.blo
blockout seg002F.zzc seg002F.blo
blockout seg0030.zzc seg0030.blo
blockout seg0031.zzc seg0031.blo
15    blockout seg0032.zzc seg0032.blo
blockout seg0033.zzc seg0033.blo
blockout seg0034.zzc seg0034.blo
blockout seg0035.zzc seg0035.blo
blockout seg0036.zzc seg0036.blo
20    blockout seg0037.zzc seg0037.blo

rem ************************************************************

25    rem
rem CONDSEG.BAT
rem
rem This batch file makes a call to condense for each of the segment files
rem generated by GENSEGS
30    rem
rem
condense seg0000.zzz FF seg0000.con seg0000.zzc
condense seg0001.zzz FF seg0001.con seg0001.zzc
35    condense seg0002.zzz FF seg0002.con seg0002.zzc
condense seg0003.zzz FF seg0003.con seg0003.zzc
condense seg0004.zzz FF seg0004.con seg0004.zzc
condense seg0005.zzz FF seg0005.con seg0005.zzc
condense seg0006.zzz FF seg0006.con seg0006.zzc
40    condense seg0007.zzz FF seg0007.con seg0007.zzc
condense seg0008.zzz FF seg0008.con seg0008.zzc
condense seg0009.zzz FF seg0009.con seg0009.zzc
condense seg000A.zzz FF seg000A.con seg000A.zzc
condense seg000B.zzz FF seg000B.con seg000B.zzc
45    condense seg000C.zzz FF seg000C.con seg000C.zzc
condense seg000D.zzz FF seg000D.con seg000D.zzc
condense seg000E.zzz FF seg000E.con seg000E.zzc
condense seg000F.zzz FF seg000F.con seg000F.zzc
condense seg0010.zzz FF seg0010.con seg0010.zzc
50    condense seg0011.zzz FF seg0011.con seg0011.zzc
condense seg0012.zzz FF seg0012.con seg0012.zzc
condense seg0013.zzz FF seg0013.con seg0013.zzc
condense seg0014.zzz FF seg0014.con seg0014.zzc
condense seg0015.zzz FF seg0015.con seg0015.zzc
55    condense seg0016.zzz FF seg0016.con seg0016.zzc
condense seg0017.zzz FF seg0017.con seg0017.zzc
condense seg0018.zzz FF seg0018.con seg0018.zzc
condense seg0019.zzz FF seg0019.con seg0019.zzc
condense seg001A.zzz FF seg001A.con seg001A.zzc
60    condense seg001B.zzz FF seg001B.con seg001B.zzc
condense seg001C.zzz FF seg001C.con seg001C.zzc
condense seg001D.zzz FF seg001D.con seg001D.zzc
condense seg001E.zzz FF seg001E.con seg001E.zzc
condense seg001F.zzz FF seg001F.con seg001F.zzc
condense seg0020.zzz FF seg0020.con seg0020.zzc
condense seg0021.zzz FF seg0021.con seg0021.zzc
condense seg0022.zzz FF seg0022.con seg0022.zzc
condense seg0023.zzz FF seg0023.con seg0023.zzc
condense seg0024.zzz FF seg0024.con seg0024.zzc
condense seg0025.zzz FF seg0025.con seg0025.zzc
condense seg0026.zzz FF seg0026.con seg0026.zzc
condense seg0027.zzz FF seg0027.con seg0027.zzc
condense seg0028.zzz FF seg0028.con seg0028.zzc
condense seg0029.zzz FF seg0029.con seg0029.zzc
condense seg002A.zzz FF seg002A.con seg002A.zzc
condense seg002B.zzz FF seg002B.con seg002B.zzc
condense seg002C.zzz FF seg002C.con seg002C.zzc
condense seg002D.zzz FF seg002D.con seg002D.zzc
condense seg002E.zzz FF seg002E.con seg002E.zzc
condense seg002F.zzz FF seg002F.con seg002F.zzc
condense seg0030.zzz FF seg0030.con seg0030.zzc
condense seg0031.zzz FF seg0031.con seg0031.zzc
condense seg0032.zzz FF seg0032.con seg0032.zzc
condense seg0033.zzz FF seg0033.con seg0033.zzc
condense seg0034.zzz FF seg0034.con seg0034.zzc
condense seg0035.zzz FF seg0035.con seg0035.zzc
condense seg0036.zzz FF seg0036.con seg0036.zzc
condense seg0037.zzz FF seg0037.con seg0037.zzc

rem ************************************************************

30  
rem  
rem GROUPOUT.BAT  
rem  
rem The batchfile group the output of the condense and blockout stages
rem into one file that can be processed by TAGINC to set the tag field
rem to be the same as the 2 HEX character before the . in the filename.
rem The last character of the input file to TAGINC will be changes to a P
rem to indicate Processed, and this will be used as the output file.
rem

40  
rem  
copy seg0000.con + seg0000.blo seg0000.zzc >nul

taginc seg0000.zzc foo -N
copy seg0001.con + seg0001.blo seg0001.zzc >nul

taginc seg0001.zzc foo -N
45  
copy seg0002.con + seg0002.blo seg0002.zzc >nul

taginc seg0002.zzc foo -N
copy seg0003.con + seg0003.blo seg0003.zzc >nul

taginc seg0003.zzc foo -N
copy seg0004.con + seg0004.blo seg0004.zzc >nul

taginc seg0004.zzc foo -N
copy seg0005.con + seg0005.blo seg0005.zzc >nul

taginc seg0005.zzc foo -N
copy seg0006.con + seg0006.blo seg0006.zzc >nul

taginc seg0006.zzc foo -N
50  
copy seg0007.con + seg0007.blo seg0007.zzc >nul

taginc seg0007.zzc foo -N
copy seg0008.con + seg0008.blo seg0008.zzc >nul

taginc seg0008.zzc foo -N
copy seg0009.con + seg0009.blo seg0009.zzc >nul

taginc seg0009.zzc foo -N
copy seg000A.con + seg000A.blo seg000A.zzc >nul

taginc seg000A.zzc foo -N
55  
47
copy seg002A.con + seg002A.zxc seg002A.zcc >nul
taginc seg002A.zcc foo -N
copy seg002B.con + seg002B.zxc seg002B.zcc >nul
taginc seg002B.zcc foo -N
copy seg002C.con + seg002C.zxc seg002C.zcc >nul
taginc seg002C.zcc foo -N
copy seg002D.con + seg002D.zxc seg002D.zcc >nul
taginc seg002D.zcc foo -N
copy seg002E.con + seg002E.zxc seg002E.zcc >nul
taginc seg002E.zcc foo -N
copy seg002F.con + seg002F.zxc seg002F.zcc >nul
taginc seg002F.zcc foo -N
copy seg0030.con + seg0030.zxc seg0030.zcc >nul
taginc seg0030.zcc foo -N
copy seg0031.con + seg0031.zxc seg0031.zcc >nul
taginc seg0031.zcc foo -N
copy seg0032.con + seg0032.zxc seg0032.zcc >nul
taginc seg0032.zcc foo -N
copy seg0033.con + seg0033.zxc seg0033.zcc >nul
taginc seg0033.zcc foo -N
copy seg0034.con + seg0034.zxc seg0034.zcc >nul
taginc seg0034.zcc foo -N
copy seg0035.con + seg0035.zxc seg0035.zcc >nul
taginc seg0035.zcc foo -N
copy seg0036.con + seg0036.zxc seg0036.zcc >nul
taginc seg0036.zcc foo -N
copy seg0037.con + seg0037.zxc seg0037.zcc >nul
taginc seg0037.zcc foo -N

rem *****HEX2CDL.BAT**********************************************

@echo off
REM --TEST THE INPUT PARAMETERS --
35 IF %1NOTHING=NOTHING GOTO USAGE
IF %2NOTHING=NOTHING GOTO USAGE
rem Now clean up from before
if exist templ.hex del templ.hex
40 if exist chkmsg.msg del chkmsg.msg
for %%f in (*.zp) do del %%f
for %%f in (*.zxc) do del %%f
for %%f in (*.con) do del %%f
for %%f in (*.blo) do del %%f
45 for %%f in (*.zzz) do del %%f
rem First convert to hex with width of 6
convert %1 templ.hex -N06
rem Then generate the CDL checksum messages for each code sector
cdcheck templ.hex -M > chkmsg38.msg
rem foo is ignored, TAGINC w/ _N as 3rd param generates its own filename
55 taginc chkmsg38.msg foo -N
rem Generate 55 file one for each segment of the EEPROM
rem filenames will be seg0000.zzz thru seg0037.zzz
rem the numbers in the filename are HEX indicating the segment number
gensegs templ.hex
rem Then remove the FF's from the 55 segment files

49
call condseg

rem Now remove all contiguous download blocks into messages
call blockseg

rem now I need to group the output of condseg with blockseg then change the tags
rem to indicate to the hecz which segment of EEPROM they are
call groupout

rem Now create the final output file from all the *.zzp files from taginc
rem the checksum messages head off the file
copy chkmsg38.msgP %2 >nul
for %f in (*.zzp) do type %f >> %2

rem Now clean up
if exist templ.hex del templ.hex
if exist chkmsg.msg del chkmsg.msg
for %f in (*.zzp) do del %f
for %f in (*.zzc) do del %f
for %f in (*.con) do del %f
for %f in (*.blo) do del %f
for %f in (*.zzz) do del %f

goto END
:USAGE
ECHO.
ECHO.
ECHO.
ECHO.
ECHO.
ECHO.
35 ECHO. USAGE:
ECHO.
ECHO. %0 FILE1 FILE2
ECHO.
ECHO.WHERE FILE1 = INPUT FILE: IN HEX FILE FORMAT
ECHO.
ECHO. FILE2 = OUTPUT FILE: SEGMENTED MESSAGE FILE READY TO DOWNLOAD
ECHO.
ECHO.THESE UTILITY CONVERTS A HEX FILE TO A SEGMENTED MESSAGE FILE CONTAINING A
ECHO.SERIES OF DOWNLOAD MESSAGES WITH THE TAG FIELD INDICATING THE SEGMENT
ECHO.
ECHO.
ECHO.
ECHO.
ECHO.
ECHO.

:EEND
rem this causes CR before prompt
echo .

/**** BLOCKOUT.C ****/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include 'constant.h'

50
#include "helpers.c"
#include "proto.h"

/* Function Prototypes */
long Locate(long Start, int which);
//int Hex_Value(char c);
//unsigned int Get_Address(char *Line);
//int Get_Bytes(void);
//void Dump_Data(void);
//int Generate_Hex_File(FILE *fpout, int outputBytes);
int Generate_Msg_File(FILE *fpout, int bufsize);

/*
#define ARRAY_SIZE 32768L //0xFFFF - 0x8000 Valid EEPROM Range
#define MAX_BUFSIZE 0xFF //CMI MAX Buffer size
#define MIN_BUFSIZE 0x0F //CMI MIN Buffer size
#define DEF_BUFSIZE 0xFF //CMI DEFAULT Buffer size

#define CMI_CDL_CHECKSUM 0x38 //CMI Checksum msg
#define CMI_CDL_FILL_PATT 0x39 //CMI Fill pattern msg (RLE)
#define CMI_CDL_NORM 0x3A //CMI Nomral DL message
#define CMI_CDL_START_CONT 0x3B //CMI Continuation msg started with START_CONT
#define CMI_CDL_CONT 0x37 //CMI Continuation msg started with START_CONT
#define CMI_CDL_STAT_INSTR 0x36 //CMI Download instructions/status

#define BYTES_IN_NORM_MSG 6
#define BYTES_IN_START_CONT_MSG 6
#define BYTES_IN_CONT_MSG 9
#define END_OF_Z_STRING 0
#define START_OF_Z_STRING 1

#define DATA_PER_CONT_MSG 66 //number of bytes in a continucatin message
#define DATA_START 32768L //Decima; form of 0x8000
#define LINE_LENGTH 81 //Input and output line length
#define MAX_BYTES_PER_LINE 50 //If you up this best up line_length too !

char Inline[LINE_LENGTH];
char Outline[LINE_LENGTH];
char c; //This is the character currently under evaluation

/* The [0] element of this array is the data at hex address 0x6000, which is
DATA_START (decimal) so, if I convert the 4 char Hex address to a decimal and
then subtract DATA_START I have the index to the array where the data for the
hex record is to start. The data in the array is just hex characters.
Data[address][0] holds the Most significant hex character and Data[address][1]
holds the other character.
*
*/
char huge Data[ARRAY_SIZE][2];
unsigned int tempadd;
unsigned int minadd, maxadd;

int Nbytes; //Number of data bytes on the line
unsigned int address; //Decimal version of address, remember to subtract 0x8000
                    //before using to index array
unsigned long File_Checksum;
//A 32 bit checksum will be appended to file & to screen, used by downloader
int outputBytes; //Number of data bytes on the line of the output file

/*******************************/

int Generate_Msg_File(FILE *fpout, int bufsize)
{
    /*
    This routine maximizes the use of the START_CONT_MSG and CONT_MSG messages to minimize
    the amount of messages sent to the CMI for a download.
    */
    unsigned int i,j, address;
    char done=0;
    long startB, stopB; //start and stop of a block of contiguous code.
    int Max_Bytes_Per_Block; //an int < 256
    long size; //size in bytes of the contiguous section of code
    int blocks; //a block holds the maximum # of contig bytes we can send to CMI buffer
    int Max_N_of_CONT_Msgs;
    // Max_N_of_CONT_Msgs = (size - 1) / Max_Bytes_Per_Block + 1

    // Init the startB and stopB
    startB = address;
    stopB = startB + size - 1;
    // Check if the block overlaps with...
int bytes_to_send;
int num_CONT_msgs;

Max_N_of_CONT_Msgs = (int)((bufsize - BYTES_IN_START_CONT_MSG) / BYTES_IN_CONT_MSG);
Max_Bytes_Per_Block = (Max_N_of_CONT_Msgs *BYTES_IN_CONT_MSG)+BYTES_IN_START_CONT_MSG;
i=0; startB=0,address=0;

if ( (Data[0][0] == 'Z') && (Data[0][0] == 'Z') )
{
  //started with void area, look for non-Z starting point
  startB=Locate(0,END_OF_Z_STRING);
  address = startB;
}

while ( done != 1 )
{
  //Starting at startB find the next Z that indicates the end of a continued run of data
  stopB=Locate(startB,START_OF_Z_STRING); //find end of contig data
  size = stopB - startB; //Num of contig bytes
  if ( size > 0 ) //end of array yet
  {
    address = startB;
    if ( size > BYTES_IN_NORM_MSG )
    {
      //a full block is 255 bytes of data
      blocks = (size / Max_Bytes_Per_Block); //full blocks
      //will we have a partial block ??
      if (((size - (blocks * Max_Bytes_Per_Block)) > BYTES_IN_NORM_MSG)
        blocks++;

      while ( (blocks > 0) && (size > BYTES_IN_NORM_MSG) )
      {
        blocks--;
        bytes_to_send = min(Max_Bytes_Per_Block, size);

        //first the START_CONT_MSG message
        fprintf(fpout, "%2X", CMI_CD_CONT); //msg #
        fprintf(fpout, "0000", (address-DATA_START));
        fprintf(fpout, "%02X", bytes_to_send);

        for(i=0; i < BYTES_IN_START_CONT_MSG; i++)
        {
          fprintf(fpout, "%c", Data[address][0],Data[address][1]);
          address++;
          size--;
          bytes_to_send--;
        } //end for
        fprintf(fpout, "\n");

        //now all the CONT_MSG messages
        num_CONT_msgs = (bytes_to_send /BYTES_IN_CONT_MSG);

        //are we going to need a spill over message
        if (((bytes_to_send - (num_CONT_msgs *BYTES_IN_CONT_MSG)) > 0 )
          num_CONT_msgs++;

        //limit to the max
        num_CONT_msgs = min( num_CONT_msgs, Max_N_of_CONT_Msgs);

        for (i=0; i < num_CONT_msgs; i++)
        {
          fprintf(fpout, "%2X", CMI_CD_CONT); //msg
          fprintf(fpout, "0000"); //cmi # and tag

          for (j=0; j < BYTES_IN_CONT_MSG; j++)
          {
            if ( address < stopB )
            {
              fprintf(fpout, "%c", Data[address][0],Data[address][1]);
              address++;
              size--;
            }
          }
        }
      }
    }
  }

52
} else
  fprintf(fpout,"000: //pad w/ 0's
}

} //end for
fprintf(fpout,"\n");
} //end for
}//end while (blocks < 0)

if (size > OL)
{
  //we have a case where we have less bytes than needed to start
  //a block 20
  //we will send a normal down load message
  fprintf(fpout,"%2X",CMIDLMNORM);
  fprintf(fpout,"%000%4X",(address+DATA_START));
  if (size < 0x10)
    fprintf(fpout,"%0X",size);
  else
    fprintf(fpout,"%2X",size);
  
for (i=0;i<size;i++)
{
  //dump out the data bytes
  fprintf(fpout,"%c",Data[address][0].Data[address][1]);
  address++;
}
for (i=0;i<(BYTES_IN_NORM_MSG - size);i++)
{
  //pad the data if necessary
  fprintf(fpout,"00");
}
fprintf(fpout,"\n");
}//end if
}//end if (size > TIMES_IN_NORM_MSG)

else
{
  //need to generate a CML_NORM message
  fprintf(fpout,"%2X",CMIDLMNORM);
  fprintf(fpout,"%000%4X",(address+DATA_START));
  if (size < 0x10)
    fprintf(fpout,"%0X",size);
  else
    fprintf(fpout,"%2X",size);

for (i=0;i<size;i++)
{
  //dump out the data bytes
  fprintf(fpout,"%c",Data[address][0].Data[address][1]);
  address++;
}
for (i=0;i<(BYTES_IN_NORM_MSG - size);i++)
{
  //pad the data if necessary
  fprintf(fpout,"00");
}
fprintf(fpout,"\n");
}//end else

startB = Locate(stopB, END_OF_Z_STRING); //find start of next run of contig

return(0);  //success
}//end function Generate_Msg_File

53
```c
int main(int argc, char *argv[]) {
    char c;
    long i;
    FILE *fpin;  // Input file pointer
    FILE *fpout;  // Output file pointer
    int bufsize;
    File_Checksum = 0;
    maxadd = 0;
    minadd = 0xFFF;

    if ( (argc < 3) || (argc > 4) ||
         (argc == 4) && (argv[3][0] != '-') && (argv[3][1] != 'N'))
    {
        fprintf(stderr, "USAGE: %s Input_File Output_File -Nxx\n", argv[0]);
        fprintf(stderr, "This utility will read in a partial HEX file and generate\n");
        fprintf(stderr, "Messages using CMI_CDL_START_CONT followed by several\n");
        fprintf(stderr, "CMI_CDL_CONT messages. The -Nxx option allows the user to\n");
        fprintf(stderr, "specify the CMI buffer size. The xx in the -Nxx are two HEX \n");
        fprintf(stderr, "characters that specify the number of bytes in the CMI buffer. \n");
        fprintf(stderr, "Ex: -N64 (100 bytes) By Default 256 (0xFF) byte buffer is used.\n");
        fprintf(stderr, "Range of N : 0x0F <= N <= 0xFF \n\n");
        return 1;
    }

    if ( (fpin=fopen(argv[1],"rt") ) == NULL)
    {
        fprintf(stderr, "Cannot open input file \s\n",argv[1]);
        return 1;
    }

    if ( (fpout=fopen(argv[2],"wt") ) == NULL)
    {
        fprintf(stderr, "Cannot open output file \s\n",argv[2]);
        return 1;
    }

    for (i=0; i< ARRAY_SIZE; i++)
    {
        c='F';
        Data[i][0] = c; //remember its characters and F's are blanks
        Data[i][1] = c;
    }

    if ( (argc==4) && (argv[3][0] == '-') && (argv[3][1] == 'N') )
    {
        if ( (argc==4) && (isdigit(argv[3][2]) && isdigit(argv[3][3])) )
        {
            //the user is specifying the width
            c = argv[3][2];
            bufsize = 16 * Hex_Value(c);
            c = argv[3][3];
            bufsize += Hex_Value(c);
            if (bufsize > MAX_BUFSIZE )
                {
                    fprintf(stderr, "Error: N value > %dn",MAX_BUFSIZE);
                    return 1;
                }
            if (bufsize < MIN_BUFSIZE )
                {
                    fprintf(stderr, "Error: N value < %dn",MIN_BUFSIZE);
                    return 1;
                }
        }
        else
            {
                fprintf(stderr, "You must provide a valid 2 character HEX value after -N\n");
                return 1;
            }
    }
    else
        bufsize = DEF_BUFSIZE;
}```
} //blank out the array
for (i=0; i<ARRAY_SIZE; i++)
{
    Data[i][0]='Z';
    Data[i][1]='Z';
}

while ((fgets(Inline.LINE_LENGTH-1, fpin)))
{
    if (Inline[8] != '1') //a 1 in this loc. indicates last line
    {
        //Process the input line
        if (Inline[0] != 'r')
        {
            fprintf(stderr, "Invalid Hex file %s: %s. argv[1].line[0]: %s
            return 1;
        }
        NBytes=Get_Bits(&Inline[0]);
        address = Get_Address(&Inline[0]); //starting address from HEX line
        //Now as many CHARACTERS in the input line, stuff them into the Data array
        for (i=0; i<(NBytes*2)+1; i++)
        { //each byte has 2 hex characters
            if ( address >= DATA_START ) //then we need to put it into array
                //otherwise it's ROM Code & we don't care
                if (address > maxaddr) maxaddr = address;
                if (address < minaddr) minaddr = address;
                //Data always starts at the 9th character
                c = Inline[9*i+1];
                tempadd = address + DATA_START; //translate, ie O0800 -> 00000
                Data[tempadd][0] = c; //Get the MS(Most signif) Character
                Data[tempadd][1] = c;
        } //end if
        address++; /*when we have both characters */
    } //end for
} //end while

Generate_Hex_File(fpout, OutputBytes);
Generate_Msg_File(fpout, bufsize);
fclose(fpin);
fclose(fpout);
printf(".*");
return 0; /* indicate success to the world */
} //end main

/*****************************************************************************/

long Locate(long Start, int which)
{
    /* Either the address of the ZZ will be returned or ARRAY_SIZE will
    be indicating NOT found.
    //if which = 1, start at Start+1 and look for next occurrence of ZZ
    //if which = 0, start at Start+1 and keep looking until it's NOT a ZZ
    Start++; 
    if (which == 1)
    { 
        while (Start < ARRAY_SIZE)
            { 
                if ((Data[Start][0] == 'Z') && (Data[Start][1] == 'Z'))
                    { 
                        return (Start);
                } 
        } 
    } 
}
```c
Start++;
} //end while

if (which == 0)
{
    while ( (Start < ARRAY_SIZE) &&
            (Data[Start][0] == 'Z') &&
            (Data[Start][1] == 'Z'))
    {
        Start++;
    } //end while

    return (Start);
} //end if

return (ARRAY_SIZE);
}

/**************************************************************************
**************************************************************************/

/*** CDLCHECK.C ***/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "constant.h"
#include "helpers.c"
#include "proto.h"

/* Function Prototypes */
int Hex_Value(char c);
unsigned int Get_Address(char *Line);
int Get_Bytes(void);

char huge Data[ARRAY_SIZE][2];
char Inline[LINE_LENGTH];
char Outline[LINE_LENGTH];
char buf1[600]; //256 bytes = 512 nibbles (hex characters) with extra

char c; //This is the character currently under evaluation
unsigned long checksum; //must be a 32 bit checksum
unsigned int tempadd;
unsigned int minadd, maxadd;
char OK;

int NBytes; //Number of data bytes on the line
unsigned int address; //Decimal version of address, remember to subtract 0x8000 before
                       //using to index array

void Usage(char *Str)
{
    fprintf(stderr, "USAGE: %s Input_File [-X][-M][-N] \n", Str);
    fprintf(stderr, "This utility will generate 56 checksums for the EEPROM \n");
    fprintf(stderr, "of the hex file specified by Input_File. The values are in decimal \nformat:\n");
    fprintf(stderr, "and are one per line by default. The -X is optional, but if provided \nwill\n");
    fprintf(stderr, "display the checksums in HEX format one on each line. The -M option will\n");
    fprintf(stderr, "generate a series of download messages that can be used to download the.\n");
    fprintf(stderr, "generate checksums to the CHI. You may use -X, or -M or Neither,\n");
    fprintf(stderr, "not both.\n");
}

int main(int argc, char *argv[])
{
    char c;
    long i;
    long j;
    long k;
    long sector_checksum[57];
    long foo;
    int a;
```
int buf_ptr;
int temp;
unsigned int address;
unsigned char checksum;

FILE *fpin;  //Input file pointer
FILE *fput;  //Output file pointer

maxadd = 0;
minadd = 0xFFFF;
checksum=0;

if ( argc > 3 )
{
    Usage (argv[0]);
    return 1;
}

OK=0;
if ( (argc == 3) && (strcmp(argv[2],"-X") == 0) )
    OK =1;
if ( (argc == 3) && (strcmp(argv[2],"-M") == 0) )
    OK =1;
if ( (argc == 3) && (strcmp(argv[2],"-H") == 0) )
    OK =1;

if ( argc == 2) //just the two input files, def output is decimal
    OK = 1;

if (OK == 0)
{
    Usage (argv[0]);
    return 1;
}

if ( (fpin=fopen(argv[1],"rt")) == NULL)
{
    fprintf(stderr, "Cannot open input file \$s\",argv[1], ".\n");
    return 1;
}

//Read in the input HEX file
while((fgets(Inline, LINE_LENGTH-1,fpin)))
{
    if (Inline[8] != '1') //a 1 in this loc. indicates last line
    {
        //Process the input line
        if (Inline[0] == ':')
        {
            fprintf(stderr,"Invalid Hex file \$s\: \s, \n",argv[1],Inline[0]);
            return 1;
        }
        Nbytes=Get_Bytes(&Inline[0]);
        address = Get_Address(&Inline[0]);  //starting address from HEX line

        //Now for as many CHARACTERS in the input line, stuff them into the Data array
        for (i=0; i<(Nbytes*2); i+=2) //each byte has 2 hex characters
        {
            if ( (address <= DATA_START) && (address <= (long)MAX_CHECKSUM_ADDR ) )
                //then we need to put it into array
                //otherwise it's ROM Code or DATA Page (> 0xF000) & we don't care
                {
                    //Data always starts at the 9th character
                    c = Inline[9+i];
                    tempadd=address-DATA_START;  //translate, ie 0x8000 -> 0x0000
                    Data[tempadd][0] = c;  //Get the MSB(Most signif) Character
                    c = Inline[9+i+1];  //Get the LSB(Least Sig) Character
                    Data[tempadd][1] = c;
                    //fprintf(fput,"%X, %X\n",address, checksum);
                }  //end if
            address++;  //when we have both characters */
} // end for
} //end if
} //end while

//Clear out the checksums
for (i=0; i<57; i++)
{
    sector_checksum[i] = 0;
}

#define KK 56 //the 57th (array location 56) sector is the checksum of the checksums

//Now generate the sector checksums
for (i=0; i<KK; i++)
{
    j = i * 512; //each sector is 512 bytes
    for (k=0; k<512; k++)
    {
        c=Data[j][0];
        temp = (Hex_Value(c)<<4);
        c=Data[j][1];
        temp += Hex_Value(c);
        sector_checksum[1] += temp;
        /*
        printf("j=%d",j);
        printf("c=0x%02c\n",Data[j][0]);
        printf("c=0x%02c\n",Data[j][1]);
        printf(\"%02X\n\",sector_checksum[1]);
        */
        j++;
    }
}

//now for the checksum of the checksums
for (i=0; i<KK; i++)
{
    //we wanted a bitwise checksum !!!
    foo = (sector_checksum[i] & 0xFF000000); //Mask off MSR
    k=foo>>24;
    sector_checksum[KK] += k;
    foo = (sector_checksum[i] & 0x00FF0000);
    k=foo>>16;
    sector_checksum[KK] += k;
    foo = (sector_checksum[i] & 0x0000FF00);
    k=foo>>8;
    sector_checksum[KK] += k;
    foo = sector_checksum[i] & 0x000000FF;
    sector_checksum[KK] += foo;
}

//We will now generate the user specified output format desired
if ( (argc == 3) && strcmp(argv[2],"-X") == 0 )
{
    for (i=0; i<57; i++)
    {
        //NOTE !!! It took a LONG TIME to figure out that when displaying
        //a long you MUST have it in a SEPARATE printf !!!!
        printf("Sector %d checksum = ", i);
        printf("%08X \n",sector_checksum[i]);
    }
}

if ( (argc == 3) && strcmp(argv[2],"-H") == 0 )
{
I will generate a series of output hex formatted data lines to the output file. Each of the checksums will be on their own hex output line.

And of course to make things a little more interesting each of these output lines must have a checksum calculated for it...that is the variable checksum.

address = 0x0000;  //Starting location for the checksum data
for (i=0; i<57; i++) //56 * 8 characters per Checksum = 448 HEX Chars = 224 bytes
{
  //printf("%08X",sector_checksum[i]);
  sprintf(&buf1[0],">04");
  sprintf(&buf1[3],">04",address);
  sprintf(&buf1[7],">00");
  sprintf(&buf1[9],">081X",sector_checksum[i]);
  //calculate byte wise checksum for output line
  checksum = 0;
  for ( j = 1; j<17; j+=2 ) //
  {
    checksum += (Hex_Value(buf1[j])<<4) ; // the MSB
    checksum += (Hex_Value(buf1[j+1]));  // the LSB
  }
  checksum = ~checksum + 1; //get the 1's comp
  checksum = checksum & 0xFF; // we only want this much
  sprintf(&buf1[17],">02X",checksum);    //insert checksum into char string
  printf("%s\n",buf1); //barf it out
  address += 4; //each checksum is a long... 4 bytes... next checksum please...
}

//end of Hex output file processing

if ( argc == 3 ) && (strcmp(argv[2],"-M")) == 0 )
{
  /*
   * generate a series of download messages to load the checksums.
   * I'll use a string to hold the checksums to make preping the messages easier
   * One Download code block will do it but there will need to be padding. The DL block will contain 224 bytes (56 longs * 4 bytes per long) = 224 bytes.
   * We will need to transmit 231 bytes (125 * 9) + 6 so we will need to pad
   * 231-224 = 7 bytes
   */
  for (i=0; i<57; i++) //56 * 8 characters per Checksum = 448 HEX Chars = 224 bytes
  {
    //printf("%08X",sector_checksum[i]);
    sprintf(&buf1[1*8],">081X",sector_checksum[i]);
  }

  printf("380000F0000E814");  //CMU_CDLS_1NT address = F000. 228 bytes (57 * 4)
  buff_ptr =0;  //which means the last byte of the last 37 message is padded
  for (i=0;i<12;i++)
  {
    printf("%c",buf1[buff_ptr]); //first 6 bytes of data from out buffer
    buff_ptr++;
  }
  printf("\n");
  for (i=0; i<25; i++) //25 0x37 messages please and thanks
  {
    printf("370000");
    for (j=0;j<18;j++)
    {
      printf("%c",buf1[buff_ptr]); //9 bytes of data from out buffer
      buff_ptr++;
    }
    printf("\n"); //end of the line
  }
}
if (argc == 2)
for (i=0; i<57; i++)
{
    printf("sector_checksum[%d] = ",i);
    printf("\n",sector_checksum[i]);
}
}
fclose(fpin);
return 0; /* indicate success to the world */

/***********************/
/***********************/

/*** CONDENSE.C ***/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "constant.h"
#include "helpers.c"
#include "proto.h"

/* Function Prototypes */
int Lines_In_File(FILE *fp);
int Hex_Value(char c);
unsigned int Get_Address(char *Line);
int Get_Bytes(char *Line);
void Dump_Data(void);
char Get_Valid_Hex_Record (FILE *fp, char *Line);
void dump_message_to_file(FILE *fpout, int FF_Count, int FF_Start.Addr, char *Str);
void Display_Condense_Usage(char *);  // Displays usage information to user

/*
#define ARRAY_SIZE 32768L //0xFFFF = 0x8000 Valid EEPROM Range
#define DATA_START 32768L // Decimal form of 0x8000
#define LINE_LENGTH 150  // Input and output line length
#define TRUE 0
#define FALSE 1
#define MAX_CMD_BUFFSIZE 256

#define CM1_CDL_FILL_PATT 0x39
*/
char huge Data[ARRAY_SIZE][2];
char Inline[LINE_LENGTH];  // We will look at two lines at a time
char Inline2[LINE_LENGTH];  // We will look at two lines at a time
unsigned int address1;  // each line has an address
unsigned int address2;  // each line has an address
char Outline[LINE_LENGTH];  //used to build up an output line
char c;  // This is the character currently under evaluation
char Search_Str[12];  // & occurrences of the Search character from command line
int Nbytes1;  // Number of data bytes on the line
int Nbytes2;  // Number of data bytes on the line

// These are the input argument positions in argv[]
#define INPUT_FILE_ARG 1
#define HEX_STRING_ARG 2
#define MSG_OUTPUT_FILE_ARG 3
#define DATA_OUTPUT_FILE 4
// Other Related input constants
#define REQUIRED_ARGS 5
#define REQUIRED_BYTES_PER_LINE 6

/***********************/
main(int argc, char *argv[])
{
    long i;
    FILE *fpin;  // Input file pointer
    FILE *fpout1;  // Output file pointer
    FILE *fpout2;  // Output file pointer
    int doing_a_block;
    int FF_Count;
    int FF_Start_ADDR;
    ...
int InputLines;
int Rtn;
unsigned int Last_line_addr, Current_line_addr;
int pattern_found;
unsigned int bytes_written; //debug stuff
unsigned int bytes_read;
unsigned int bytes_bypassed;
unsigned int bytes_msg;

bytes_msg=0;
bytes_written=0;
bytes_read=0;
bytes_bypassed=0;

if ( argc != REQUIRED_ARGS )
{
    fprintf(stderr,"ERROR : Invalid Number of Arguments...
"
Display_Condense_Usage(argv[0]);
return 1;
}

if ((strlen(argv[HEX_STRING_ARG]) ! = 2) ||
    !isxdigit(argv[HEX_STRING_ARG][0]) ||
    !isxdigit(argv[HEX_STRING_ARG][1]))
{
    fprintf(stderr,"ERROR : 'Hex String' must be 2 Valid Hex characters describing the

data byte\n");  
    fprintf(stderr,"ERROR : 'Ex: FF\n\n");  
    Display_Condense_Usage(argv[0]);
    return 1;
}

for (i=0; i<12; i++)
{
    Search_Str[i]= argv[HEX_STRING_ARG][i]; //This is the pattern of 6 bytes
    Search_Str[i+1]= argv[HEX_STRING_ARG][i+1]; //that we will look for in each input line
}

if ((fpin=fopen(argv[INPUT_FILE_ARG],"rt"))
    == NULL)
{
    fprintf(stderr, "Cannot open input file %s", argv[INPUT_FILE_ARG], "\n\n");  
    Display_Condense_Usage(argv[0]);
    return 1;
}

if ((fpout1=fopen(argv[MSG_OUTPUT_FILE_ARG],"wt"))
    == NULL)
{
    fprintf(stderr, "Cannot open %s as output.*,argv[MSG_OUTPUT_FILE_ARG],".\n\n");  
    Display_Condense_Usage(argv[0]);
    return 1;
}

if ((fpout2=fopen(argv[DATA_OUTPUT_FILE],"wt"))
    == NULL)
{
    fprintf(stderr, "Cannot open %s as output.*,argv[DATA_OUTPUT_FILE],".\n\n");  
    Display_Condense_Usage(argv[0]);
    return 1;
}

for (i=0; i<LINE_LENGTH; i++)
{
    Inline[i] = 0;
}

Last_line_addr = (0x8000 - 6); //force first line to be contig
Doing_a_Block = FALSE;
FF_Count = 0;
FF_Start_Addr = NULL;
InputLines = Lines_In_File(fpin);
for (i=0; i < InputLines; i++) // process each line in the input file
{
    Rtn = Get_Valid_Hex_Record (fpin, &InLine[0]); // read a line from input file
    bytes_read += Get_Bytes(&InLine[0]); // debug
    if ( Rtn == 2 )
    {
        // This should not happen check for EOF
        fprintf(stderr,"Unexpected EOF encountered..terminating\n");
        exit(1);
    }
    if ( Rtn == 1 ) // this is either invalid hex or a hex line with > 6 data bytes
    {
        // just pass it on to the catch all file
        fputs(&InLine[0],fpout2);
        bytes_bypassed += Get_Bytes(&InLine[0]); // debug
        bytes_written += Get_Bytes(&InLine[0]); // debug
        continue; // start the for loop over again
    }
    Current_line_addr=Get_Address(&InLine[0]);
    // Check string just read in looking for our pattern
    // data always starts at the 9th character for a hex record
    pattern_found=strcmpi(Search_Str, &InLine[9], sizeof(Search_Str));
    if (!pattern_found && 6 == Current_line_addr) // lines are contiguous
    {
        if (pattern_found==0) // we found the pattern
        {
            if (Doing_a_Block == TRUE)
            {
                // will our buffer be full
                if ( FF_Count + 6 > MAX_CMT_BUFFSIZE )
                {
                    // we can't squeeze any more in time to dump & need to empty
                    dump_message_to_file(fpout1,FF_Count,FF_Start_Addr,Search_Str);
                    bytes_written+=FF_Count; // debug
                    bytes_msg+=FF_Count; // debug
                    FF_Count = 6;
                    FF_Start_Addr = Current_line_addr;
                    // Doing_a_Block is STILL TRUE
                }
                else
                {
                    FF_Count +=6; // just up the count
                }
            }
            else // we were NOT Doing_a_Block But now have at least 6 bytes of pattern
            {
                // while it is possible that we may only have a block of only one line
                // it will generate 1 message either way...so we'll generate this one
                FF_Count =6;
                FF_Start_Addr = Current_line_addr;
                Doing_a_Block = TRUE;
            }
        }
    } // Pattern not found
    else // we're all done with that block
    {
        dump_message_to_file(fpout1,FF_Count,FF_Start_Addr,Search_Str); // send
to file
        bytes_written+=FF_Count; // debug
        bytes_msg+=FF_Count; // debug
        FF_Count = 0; // reset our vars
        FF_Start_Addr = 0;
        Doing_a_Block = FALSE;
    }
    // just pass it through it's not what we are looking for
    fputs(&InLine[0],fpout2);
    bytes_written+=6; // debug
}

else // not contiguous to last
{

    if (Doing_a_Block == TRUE )
    {
        dump_message_to_file(fpout1, FF_Count, FF_Start_Addr, Search_Str);
        bytes_written+=FF_Count; //debug
        FF_Count = 0;
        FF_Start_Addr = 0;
        Doing_a_Block = FALSE;
    } // end if

    if (pattern_found==0) // so were not contig, we found our pattern
    {
        // while it is possible that we may only have a block of only one line
        // it will generate 1 message either way...so what the heck
        FF_Count = 6;
        FF_Start_Addr = Current_line_addr;
        Doing_a_Block = TRUE;
    } else
    {
        fputs(&Inline1[0], fpout2); //dump
        bytes_written+=6; //debug
    }
}

} // end else not contig

// gear up for next line
Last_line_addr = Current_line_addr;
} // end for

// flush the buffer
if (Doing_a_Block == TRUE )
{
    dump_message_to_file(fpout1, FF_Count, FF_Start_Addr, Search_Str);
    bytes_written+=FF_Count; //debug
    FF_Count = 0;
    FF_Start_Addr = 0;
    Doing_a_Block = FALSE;
} // end if

*/
printf("bytes_written = %u\n", bytes_written);
printf("bytes_read = %u\n", bytes_read);
printf("bytes_bypassed = %u\n", bytes_bypassed);
*/
printf(".\n");
fclose(fpin);
fclose(fpout1);
fclose(fpout2);
return 0; /* indicate success to the world */
} // end main

******************************************************************************

    char Get_Valid_Hex_Record (FILE *fp, char *Line)
{

        /*
        Input: fp : pointer to input file to read
                Line: array of chars to read data into
        Output: 0 if successful
                1 if failed tests
                2 EOF
        */

        int Nbytes;

        // read a line, if end of file, return fail status
        if (fgets(Line, LINE_LENGTH-1, fp) == NULL ) return (2);
if (Line[0] != ":")  //Hex lines MUST start with : (cept for 1st)
{
    return(1);  //bad
}
}
Nbytes=Get_Bytes(Line[0]);  //number of databytes in this line
if (Nbytes != REQUIRED_BYTES_PER_LINE)
{
    return(1);  //bad
}
return (0);  //success

//end function Get_Valid_Hex_Record

/**
 * 
 * void Display_Condense_Useage(char *Str)
 * 
 * 
 * void dump_message_to_file(FILE *fpoutl, int FF_Count, int FF_Start_Adr, char *Str)
 */

/***** CONVERT.C *****
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "constant.h"
#include "helpers.c"
#include "proto.h"

char InLine[LINK_LENGTH];
char Outline[LINE_LENGTH];
char c;  //This is the character currently under evaluation

/* The [0] element of this array is the data at hex address 0x6000, which is
DATA_START (decimal) so, If I convert the 4 char Hex address to a decimal and
then subtract DATA_START I have the index to the array where the data for the
hex record is to start. The data in the array is just hex characters.
Data[address][0] holds the Most significant hex character and Data[address][1]
holds the other character.*/
}
unsigned int tempadd;
unsigned int minadd, maxadd;
int Nbytes; // Number of data bytes on the line
unsigned int addr; // Decimal version of address, remember to subtract 0x8000
// before using to index array

unsigned long File_Checksum;
// A 32 bit checksum will be appended to file & to screen, used by downloader
int OutputBytes; // Number of data bytes on the line of the output file

int main(int argc, char *argv[])
{
    char c;
    long i;
    FILE *fpin; // Input file pointer
    FILE *fpout; // Output file pointer
    int supress; // suppress the printing of the end of file statement if = 1

    File_Checksum = 0;
    maxadd = 0;
    minadd = 0x0FFF;
    supress = 0; // by default we will print the end of file line

    if ( argc < 3 )
        fprintf(stderr, "Usage: %s Input_File Output_File -Nxx
", argv[0]);
    if (argc == 4 && (argv[3][0] == '-' && (argv[3][1] != 'N')))
        fprintf(stderr,"\n");
    if (argc == 4 && (argv[3][0] == '-') && (argv[3][1] != 'N'))
        fprintf(stderr,"\n");

    if (fpin=fopen(argv[1],"rt") == NULL)
        fprintf(stderr, "Cannot open input file \s\n", argv[1]);
    return 1;

    if (fpout=fopen(argv[2],"wt") == NULL)
        fprintf(stderr, "Cannot open output file \s\n", argv[2]);
    return 1;

    for (i=0; i<ARRAY_SIZE; i++)
    {
        c='F';
        Data[i][0] = c; // remember it's characters and F's are blanks
        Data[i][1] = c;
    }

    if ( (argc==4) && (argv[3][0] == '-') && (argv[3][1] == 'N') )
    {
        if ( (argc==4) && (isxdigit(argv[3][2]) && isxdigit(argv[3][3])) )
        {
            // the user is specifying the width
            c = argv[3][2];
            OutputBytes = 16 * Hex_Value(c);
            c = argv[3][3];
            OutputBytes += Hex_Value(c);
            if (OutputBytes > MAX_BITS_PER_LINE )
```c
{    fprintf(stderr, "Error: N value > %d\n", MAX_BYTES_PER_LINE);
    return 1;
}
} else
{    fprintf(stderr, "You must provide a valid 2 character HEX value after -N\n");
    return 1;
}
} else
{    OutputBytes = DEFAULT_DATA_WIDTH;
}
if (argc==4) && (argv[3][0] == '-') && (argv[3][1] == 'H')
{    if (argc==4) && (isxdigit(argv[3][2]) && isxdigit(argv[3][3]))
    {        //the user is specifying the width suppress = 1; //don't print end of file indicator
            c = argv[3][2];
            OutputBytes += 16 * Hex_Value(c);
            c = argv[3][3];
            OutputBytes += Hex_Value(c);
            if (OutputBytes > MAX_BYTES_PER_LINE) 
            {                fprintf(stderr, "Error: N value > %d\n", MAX_BYTES_PER_LINE);
                return 1;
            }
   }
} else
{    fprintf(stderr, "You must provide a valid 2 character HEX value after -H\n");
    return 1;
}
}
else
{    OutputBytes = DEFAULT_DATA_WIDTH;
}
while(!fgets(Inline,LINE_LENGTH-1,fpin))
{    if (Inline[0] != '\n') //a 1 in this loc. indicates last line
    {        //Process the input line
                if (Inline[0] != '\n')
                {                    fprintf(stderr,"Invalid Hex file %s: %s, \n", argv[1], Inline[0]);
                        return 1;
                }
            NBytes = GetBytes(&Inline[0]);
            address = Get_Address(&Inline[0]);  //starting address from HEX line
            //Now for as many CHARACTERS in the input line, stuff them into the data array
            for (i=0; i<NBytes*2; i++) //each byte has 2 hex characters
            {                if ((address >= DATA_START) && address <= MAX_CHECKSUM_ADDR )
                        //then we need to put it into array
                        //otherwise it's ROM or data Code & we dont care
                                if (address > maxaddr) maxaddr = address;
                                if (address < minaddr) minaddr = address;
                                //Data always starts at the 9th character
                                c = Inline[9+i];
                                tempadd = address-DATA_START;  //translate, ie 0x8000 -> 0x0000
                                Data[tempadd][0] = c; //Get the MS(Most signif) Character
                                c = Inline[9+i+1]; //Get the LS(Least sig) Character
                                Data[tempadd][1] = c;
            }
        }
```
address++; /* when we have both characters */

} //end for

} //end if
} //end while

Generate_Hex_File(fpout, OutputBytes, suppress);

/****************************************************************************
 printf("minaddr = \ud\n",minaddr);
 printf("maxaddr = \ud\n",maxaddr);
*/
 printf(".*.");
 fclose(fpin);
 fclose(fpout);
 return 0; /* indicate success to the world */
}//end main

******************************************************************************/

******************************************************************************/

/**** GENSEGS.C ****/
#include <stdio.h>
#include <stdlib.h>
#include "string.h"
#include "constant.h"
#include "helpers.c"
#include "proto.h"

char Inline[LINT_LENGTH];
char Outline[LINT_LENGTH];
char filename[20];
char c; //This is the character currently under evaluation

/****************************************************************************
 The [0] element of this array is the data at hex address 0x6000, which is
 DATA_START (decimal) so, If I convert the 4 char Hex address to a decimal and
 then subtract DATA_START I have the index to the array where the data for the
 hex record is to start. The data in the array is just hex characters.
 Data[address][0] holds the Most significant hex character and Data[address][1]
 holds the other character.
*/
char huge Data[ARRAY_SIZE][2];
unsigned int tempadd;
unsigned int minadd,maxadd;
int Nbytes; //Number of data bytes on the line
unsigned int address; //Decimal version of address, remember to subtract 0x8000
                        //before using to index array
unsigned long File_Checksum;
//a 32 bit checksum will be appended to file & to screen, used by downloader
int OutputBytes; //Number of data bytes on the line of the output file

int main(int argc, char *argv[])
{
    char c;
    long i;
    FILE *fpin;  //Input file pointer
    FILE *fpout;  //Output file pointer

    File_Checksum = 0;
    maxadd = 0;
    minadd = 0xFFFF;

    if ( (argc != 2) )
    {
        fprintf(stderr,"USAGE: &s Input_File\n", argv[0]);
        fprintf(stderr,"This utility will read the input hex file and create 55 hex a\n");
        fprintf(stderr,"files, one for each 512 byte 'sector' off EEPROM. These files\n");
        fprintf(stderr,"will later be post processed to build an INDEXED message file\n");
        fprintf(stderr,"which will allow the NECU to download A SPECIFIC sector \n");
        fprintf(stderr,"(1/64th of EEPROM) with minimum number of messages.\n");
        return 1;
    }

    return 0;
}
if ( (fpin=fopen(argv[1],"rt")) == NULL) 
{
    fprintf(stderr, "Cannot open input file %s\n",argv[1]);
    return 1;
}
for (i=0; i< ARRAY_SIZE; i++)
{
    c='F';
    Data[i][0] = c; //remember it's characters and F's are blanks
    Data[i][1] = c;
}

while((fgets(Inline,LINE_LENGTH-1,fpin)))
{
    if (Inline[8] == '1') //a 1 in this loc. indicates last line
    {
        //Process the input line
        if (Inline[0] != ':')
        {
            fprintf(stderr,"Invalid Hex file %s: %s, \n",argv[1],Inline[0]);
            return 1;
        }
        Nbytes=Get_Bytes(&Inline[0]);
        address = Get_Address(&Inline[0]); //starting address from HEX line
        //
        //
        //
        //
        if (((address > 0x8200) && (address < 0x82FF)))
        {
            address = address; //debug hook
        }
        //Now for as many CHARACTERS in the input line, stuff them into the Data array
        for (i=0; i<Nbytes*2; i++) //each byte has 2 hex characters
        {
            if ( address >= DATA_START ) //then we need to put it into array
            {
                //otherwise it's ROM Code & we dont care
                if (address > maxadd) maxadd = address;
                if (address < minadd) minadd = address;
                //Data always starts at the 9th character
                c = Inline[9+i];
                tempadd=address-DATA_START; //translate, ie 0x8000 (000000 0000) -> 0x0000
                Data[tempadd]={0,0};//Get the MS(Most signif) Character
                c = Inline[9+i+1];//Get the LS(Least Sig) Character
                Data[tempadd][1] = c;
            }
            address++; /*when we have both characters */
        }
    }
}

for (i=0;i<56; i++) //0-55 inclusive
{
    sprintf(&file_name,"seg%04X.ZZZ",i);
    if ( (fpout=fopen(file_name,"wt")) == NULL) 
    {
        fprintf(stderr, "Cannot open output file %s\n",argv[2]);
        return 1;
    }
    Generate_Segment_Hex_File(fpout,i);
    fclose(fpout);
}
#endif //end for

printf("\n");
fclose(fpin);
return 0; /* indicate success to the world */
}//end main
```c
/**** HELPERS.C ****/

#include "constant.h"
#include "proto.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

extern char huge Data[ARRAY_SIZE][2];

/*****************************/

int Get_Bytes(char *Line)
{
    char c;
    int NBytes;
    c = Line[1];
    NBytes = 16 * Hex_Value(c);
    c = Line[2];
    NBytes += Hex_Value(c);
    return(NBytes);
} //End function Get_Bytes

/*****************************/

int Generate_ROM_Hex_File(FILE *fpout, int OutputBytes)
{
    //Address space 0x0000 => 0xFFFF ROM Space
    unsigned int i,j, tempadd;
    unsigned char checksum; //0x00 - 0xFF
    unsigned char val;
    char c;
    unsigned long File_Checksum;
    i=0;
    File_Checksum = 0;
    // while (i < ARRAY_SIZE)
    while (i < 0x5FFF) //
    {
        //Check for a short line that might happen at the end of the file
        if (i == (ARRAY_SIZE - 1))
        {
            OutputBytes = (ARRAY_SIZE - 1);
        }
        if (OutputBytes > 15)
        {
            fprintf(fpout,"%2X",OutputBytes);
            else
            {
                fprintf(fpout,"0%X",OutputBytes);
                checksum = OutputBytes;
                //Now for the address
                tempadd=i;
                //
                fprintf(fpout,"%04X",tempadd); //this will work
                val = (tempadd >> 12); // divide by 4096
                if (val < 10) c=val+48; else c=val+55; //adjust to the ASCII char 0-9=48-57, A-F=65-
                tempadd = tempadd - (val <<12);
                fputc(c,fpout);
                checksum += Hex_Value(c) * 16; //MS Nibble High
                val = (tempadd >> 8); // divide by 256
```
if (val < 10) c = val + 48; else c = val + 55; //adjust to the ASCII char

tempadd = tempadd - (val << 8);
fputc(c, fpout);
checksum += Hex_Value(c); //LS Nibble High

val = (tempadd >> 4); //divide by 16
if (val < 10) c = val + 48; else c = val + 55; //adjust to the ASCII char
tempadd = tempadd - (val << 4);
checksum += Hex_Value(c) * 16; //MS Nibble, Low

val = tempadd;
if (val < 10) c = val + 48; else c = val + 55; //adjust to the ASCII char
fputc(c, fpout);
checksum += Hex_Value(c); //LS Nibble low

//this is the record byte
fputc(’0’, fpout);
fputc(’0’, fpout);

//now for the data
for (j=0; j<OutputBytes; j++) //each line of hex holds OutputBytes data bytes
{
    c = Data[i+j][0];
    File_Checksum += Hex_Value(c) * 16; //MS Nibble
    fputc(c, fpout);
    checksum += (Hex_Value(c) << 4); //Most Sig Nibble
    c = Data[i+j][1];
    File_Checksum += Hex_Value(c); //LS Nibble
    fputc(c, fpout);
    checksum += Hex_Value(c); //Least Sig Nibble
} //end for

//now the ONE Comp of the checksum and a CR
checksum = checksum & 0xFF; //mask off all but least sig byte
checksum = 0x100 - checksum;
val = checksum >> 4 ;
if (val < 10) c = val + 48; else c = val + 55; //adjust to the ASCII char
fputc(c, fpout);
val = checksum - (val << 4);
if (val < 10) c = val + 48; else c = val + 55; //adjust to the ASCII char
fputc(c, fpout);
fputc(’\n’, fpout); //end of line
i += OutputBytes;
} //end while
fputs(‘00000001FF\n’, fpout); //indicates end of file
return(0); //success
} //end function Generate_Hex_File

/***********************************************************************/

//int Generate_Hex_File(FILE **fpout, int OutputBytes, char huge *Data[ARRAY_SIZE][2])
int Generate_Hex_File(FILE **fpout, int OutputBytes, int suppress)
{
    #define OUTPUTONLYCODEPLEASE 0x6FFF //0 maps to 0x8000, this maps to 0xEFFF, last code address

unsigned int i, j, tempadd;
unsigned char checksum; //0x00 - 0xFF
unsigned char val;
char c;
unsigned long File_Checksum;

i = 0;
File_Checksum = 0;

//while (i < ARRAY_SIZE)
while ((i < OUTPUTONLYCODEPLEASE) && OutputBytes > 15)
{
    //Check for a short line that might happen at the end of the file
    if ((ARRAY_SIZE - i) < OutputBytes)
    {
        OutputBytes = (ARRAY_SIZE - i);
    }
    if (OutputBytes > 15)
else
    fprintf(fpout,".%2X\n",OutputBytes);
    fprintf(fpout,".0%4X\n",OutputBytes);

checksum = OutputBytes;
// Now for the address
 tempadd=i+0x8000;

    fprintf(fpout, ".%04X\n",tempadd);  // this will work

//
val = (tempadd >> 12);  // divide by 4096
if (val < 10) c=val+48; else c=val+55; // adjust to the ASCII char 0-9=48-57, A-F=65-

  tempadd = tempadd - (val <<12);
  fputc(c,fpout);
  checksum += Hex_Value(c) * 16;  // MS Nibble High

val = (tempadd >> 8);  // divide by 256
if (val < 10) c=val+48; else c=val+55; // adjust to the ASCII char

  tempadd = tempadd - (val <<8);
  fputc(c,fpout);
  checksum += Hex_Value(c);  // LS Nibble High

val = (tempadd >> 4);  // divide by 16
if (val < 10) c=val+48; else c=val+55; // adjust to the ASCII char

  tempadd = tempadd - (val <<4);
  fputc(c,fpout);
  checksum += Hex_Value(c);  // MS Nibble Low

val = tempadd;
if (val < 10) c=val+48; else c=val+55; // adjust to the ASCII char

  fputc(c,fpout);
  checksum += Hex_Value(c);  // LS Nibble Low

  // this is the record byte
  fputc('0',fpout);
  fputc('0',fpout);

  // now for the data
for (j=0; j<OutputBytes; j++)  // each line of hex holds OutputBytes data bytes
{
  c = Data[i+j][0];
  File_Checksum += Hex_Value(c) * 16;  // MS Nibble
  fputc(c,fpout);
  checksum += (Hex_Value(c) << 4);  // Most Sig Nibble
  c = Data[i+j][1];
  File_Checksum += Hex_Value(c);  // LS Nibble
  fputc(c,fpout);
  checksum += Hex_Value(c);  // Least Sig Nibble
}

  // end for

  // now the ONE Comp of the checksum and a CR
checksum = checksum & 0xFF;  // mask off all but least sig byte
checksum = 0x100 - checksum;
val = checksum >> 4 ;
if (val < 10) c=val+48; else c=val+55;  // adjust to the ASCII char

  fputc(c,fpout);
  val = checksum - (val <<4);
if (val < 10) c=val+48; else c=val+55;  // adjust to the ASCII char

  fputc(c,fpout);
  fputc('\n\n',fpout);  // end of line
  i += OutputBytes;
}

// end while
if (suppress == 0)
{
    fputs(":00000001PF\n",fpout);  // indicates end of file

    // fprintf(fpout, "%X 16\n Download Checksum (Hex Decimal)\n", File_Checksum, File_Checksum);

    // fprintf(fpout, "%X 16\n Download Checksum (Hex Decimal)\n", File_Checksum, File_Checksum);

    // fclose(fpout);
    return(0);  // success
}
}  // end function Generate_Hex_File

**************************************************************************
unsigned int Get_Address(char *Line) 
{
    char c;
    unsigned int address;

    address = 0;
    c = Line[3];
    address = 4096 * Hex_Value(c);
    c = Line[4];
    address += 256 * Hex_Value(c);
    c = Line[5];
    address += 16 * Hex_Value(c);
    c = Line[6];
    address += Hex_Value(c);
    return(address);
} //End function Get_Address

/**************************************************************************/
void Dump_Data(void)
{
    long i;
    FILE *fpout;
    extern char huge Data[ARRAY_SIZE][2];

    if ((fpout = fopen("d:\tools\hex\outdata.txt", "wt"))
        == NULL)
    {
        fprintf(stderr, "Cannot open output file d:\tools\hex\outdata.txt\n");
    }

    for (i=1219; i<1319; i++)
    {
        fputc(Data[i][0],fpout);
        fputc(Data[i][1],fpout);

        if ( (i > 0) && ((i % 9) == 0) )
            fputc(\n',fpout);
    } //end for
    fclose(fpout);
} //end function Dump_data

/**************************************************************************/
int Hex_Value(char c)
{
    int Val;
    //This function returns the hex value of the input character
    //If it is not a hex character then it returns -1
    printf("Hex_Value: char is \%\n",c);
    switch ( c )
    {
    case '0':Val= 0;break;
    case '1':Val= 1;break;
    case '2':Val= 2;break;
    case '3':Val= 3;break;
    case '4':Val= 4;break;
    case '5':Val= 5;break;
    case '6':Val= 6;break;
    case '7':Val= 7;break;
    case '8':Val= 8;break;
    case '9':Val= 9;break;
    case 'A':Val= 10;break;
    case 'B':Val= 11;break;
    case 'C':Val= 12;break;
    case 'D':Val= 13;break;
    case 'E':Val= 14;break;
    case 'F':Val= 15;break;
    case 'a':Val= 10;break;
    case 'b':Val= 11;break;
    case 'c':Val= 12;break;
    case 'd':Val= 13;break;
    case 'e':Val= 14;break;
    case 'f':Val= 15;break;
    default: Val = -1;break;
    }
}
int Lines_In_File(FILE *fp)
{
    /* This function counts the number of lines in the file */
    char Line[LINE_LENGTH];
    int i;
    i=0;
    // reset the fp
    fseek(fp,0,SEEK_SET);
    while ( (fgets(Line,LINE_LENGTH-1,fp)) != NULL ) i++ ; // count the lines
    rewind(fp);
    return (i);
}

int Generate_Segment_Hex_File(FILE *fpout, int segment)
{
    // The caller should have a separate filename for each call to this
    // routine. Each file can then be processed by "condense & blockout" to
    // optimize the download.
    // This routine will not attempt to calculate the line checksum that is
    // usually output as the last byte...I do not use this information in
    // any of my tools. 00 will be output for this checksum.
    unsigned int i, j, tempadd;
    char c;
    int OutputBytes;
    OutputBytes = 6;  // 6 bytes per HEX line
    i=0;
    while ( (i < ARRAY_SIZE)  )
    {
        // Check for a short line that might happen at the end of the file
        if ( (i == 510) )
            { OutputBytes = 2;  // last line wants to have 2 bytes 85 lines of 6 then last
                if (OutputBytes > 15)  
                    fprintf(fpout,":0%X",OutputBytes);
                else
                    fprintf(fpout,":%2X",OutputBytes);
            }
        // Now for the address
        tempadd=i*(segment*512)+0x8000;
        if (segment == 1)
            { tempadd = tempadd; }
        fprintf(fpout,"\%04X",tempadd);  
        // This is the record byte
        fputc(\0",fpout);
        fputc(\0",fpout);
        // Now for the data
        for (j=0; j<OutputBytes; j++)  // each line of hex holds OutputBytes data bytes
        {  
            c = Data[tempadd-0x8000+j]\01;
            }


Having now described a few embodiments of the invention, and some modifications and variations thereto, it should be apparent to those skilled in the art that the foregoing is merely illustrative and not limiting, having been presented by the way of example only. Numerous modifications and other embodiments are within the scope of one of ordinary skill in the art and are contemplated as falling within the scope of the invention as limited only by the appended claims and equivalents thereto.
WHAT IS CLAIMED IS:

1. In a wireless microcell distribution system, a system for rapidly downloading information from a first location to a plurality of remote locations at which cable microcell integrators are located, comprising:
   means for broadcasting information to update said cable microcell integrators from said first location such that multiple cable microcell integrators are simultaneously provided with update information with no response required from the cable microcell integrators to said first location.

2. The system of Claim 1, and further including means at said first location for sending a query message to each of said cable microcell integrators separately, with said message inquiring as to the receipt of the information broadcast.

3. The system of Claim 1, and further including at said first location means for transmitting to said cable microcell integrators a request to have each of said cable microcell integrators reconfigure itself to receive broadcast messages, thereby to establish a broadcast mode for each of said cable microcell integrators.

4. The system of Claim 3 wherein the transmission of said reconfiguration request is accomplished serially on an address-by-address unicast basis.

5. The system of Claim 1, and further including at each cable microcell integrator means for ascertaining the integrity of the message broadcast from said first location, thus to provide the status
of the information broadcast to the cable microcell integrator; and means for transmitting the status of a received broadcast at a cable microcell integrator to said first location.

6. The system of Claim 5, and further including means at said first location for evaluating the status of broadcast messages from each of the associated cable microcell integrators and for re-broadcasting information back to all of said cable microcell integrators for the purpose of correcting improperly received information at a cable microcell integrator.

7. The system of Claim 6, and further including means at each of said cable microcell integrators for ascertaining which of said re-broadcast information is to replace improperly received information and for replacing the improperly received information with the appropriate re-broadcast information.

8. In a wireless microcell distribution system, a method for providing rapid updates to each of the cable microcell integrators from a head end control unit, comprising the steps of:

   requesting in a unicast mode that each of the cable microcell integrators be put in a broadcast mode;

   transmitting information to all of said cable microcell integrators simultaneously in a broadcast mode; and,

   having each of the cable microcell integrators providing a message in a unicast mode indicating the status of the receipt of the broadcast message to the head end control unit, whereby each of the cable microcell integrators is placed in a broadcast mode on a unicast basis, in which the updates are transmitted to the cable microcell integrators in a broadcast mode and in which the status of the
receipt of the broadcast information is returned to the head end interface converter in a unicast mode.

9. In a wireless microcell distribution system, a method for rapidly downloading information from a first location to a plurality of remote locations at which cable microcell integrators are located, comprising the step of:

broadcasting information to update cable microcell integrators from the first location such that multiple cable microcell integrators are simultaneously provided with update information with no response required from the cable microcell integrators to the first location.

10. The method of Claim 9, and further including at the first location, the step of sending a query message to each of the cable microcell integrators separately, with the message inquiring as to the receipt of the information broadcast.

11. The method of Claim 9, and further including at the first location the step of transmitting to said cable microcell integrators a request to have each of said cable microcell integrators reconfigure itself to receive broadcast messages, thereby to establish a broadcast mode for each of the cable microcell integrators.

12. The method of Claim 11 wherein the transmission of the reconfiguration request is accomplished serially on an address-by-address unicast basis.
13. The method of Claim 9, and further including at each cable microcell integrator the steps of ascertaining the integrity of the message broadcast from the first location, thus to provide the status of the information broadcast to the cable microcell integrator; and transmitting the status of a received broadcast at a cable microcell integrator to the first location.

14. The method of Claim 13, and further including at the first location the steps evaluating the status of broadcast messages from each of the associated cable microcell integrators and re-broadcasting information back to all of the cable microcell integrators for the purpose of correcting improperly received information at a cable microcell integrator.

15. The method of Claim 6, and further including at each of the cable microcell integrators the steps of ascertaining which of the re-broadcast information is to replace improperly received information and replacing the improperly received information with the appropriate re-broadcast information.

16. In a wireless microcell distribution system, a method for providing rapid updates to each of the cable microcell integrators from a head end control unit, comprising the steps of:

requesting in a unicast mode that each of the cable microcell integrators be put in a broadcast mode;

transmitting information to all of said cable microcell integrators simultaneously in a broadcast mode; and,

having each of the cable microcell integrators providing a message in a unicast mode indicating the status of the receipt of the broadcast message to the head end control unit.
Fig. 1
(PRIOR ART)

Fig. 2

BROADCAST REQUEST, BUT NO RESPONSE REQUIRED
Fig. 3
Fig. 4