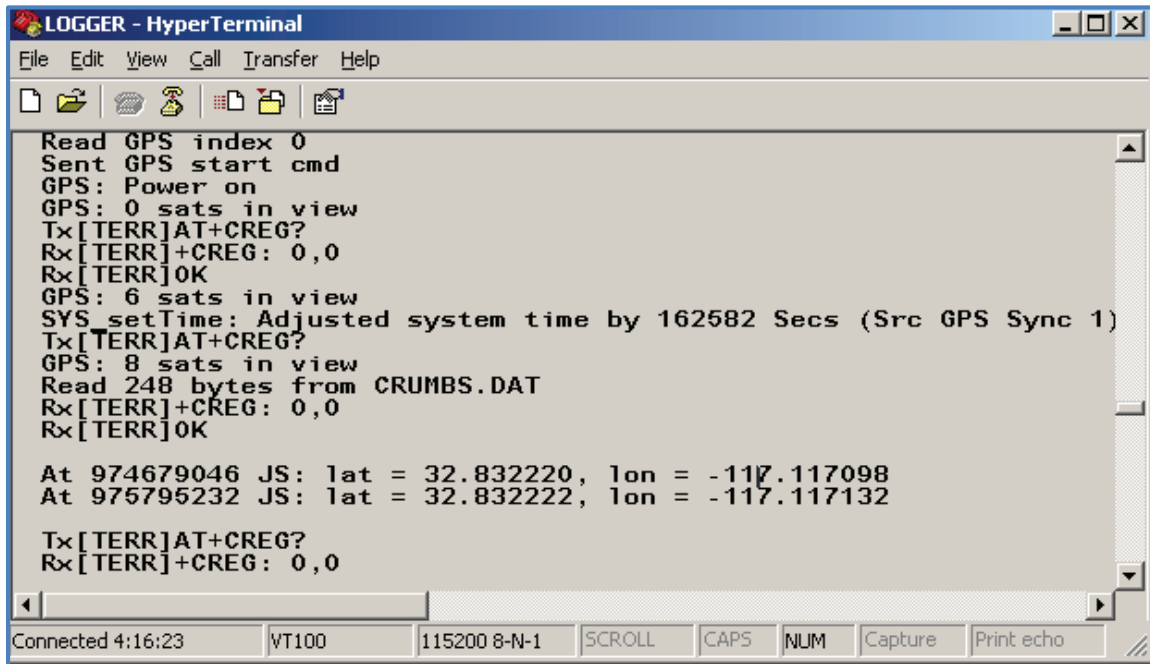


The application starts the GPS measurement on `POWER_UP` and updates an FFS file every time it receives a `POSITION_FIX` event. The file contains the last `NUM_GPS_CRUMBS` positions that were taken. Each time the modem powers on, a new location is saved in the FFS, so that a trail of positions is retained over power cycles. The first time the modem is started, the following Logger output is displayed



```

LOGGER - HyperTerminal
File Edit View Call Transfer Help

Read GPS index 0
Sent GPS start cmd
GPS: Power on
GPS: 0 sats in view
Tx[TERR]AT+CREG?
Rx[TERR]+CREG: 0,0
Rx[TERR]OK
GPS: 6 sats in view
SYS_setTime: Adjusted system time by 162582 Secs (Src GPS Sync 1)
Tx[TERR]AT+CREG?
GPS: 8 sats in view
Read 248 bytes from CRUMBS.DAT
Rx[TERR]+CREG: 0,0
Rx[TERR]OK

At 974679046 JS: lat = 32.832220, lon = -117.117098
At 975795232 JS: lat = 32.832222, lon = -117.117132

Tx[TERR]AT+CREG?
Rx[TERR]+CREG: 0,0

Connected 4:16:23 VT100 115200 8-N-1 SCROLL CAPS NUM Capture Print echo
  
```

**Figure 12-52: DemoAppFFS - Logger output for initial Power On**

It is possible to view the directory on the modem to ensure that all the files have been created and stored correctly. To do this, on the Logger port type 'd' 'f'. This gives you a menu of file utilities which can be run on NVM. It is possible to list, rename, delete and move files using these utilities.



**Note:**

See [Appendix D - Debug and utility menus](#) for more information on the debug (d) and utility (U) commands.

CONFIDENTIAL

Information classified Confidential - Do not copy (See last page for obligations)

Each time the modem is power cycled, another line is added to the position output, until the maximum number of lines is reached. After this, the oldest entry in the array is overwritten by a new position.

```

Rx[TERR]OK
Tx[TERR]AT+CGREG?
Rx[TERR]+CGREG: 0,1
Rx[TERR]OK
GPS: 5 sats in view
Tx[TERR]AT+CREG?
Rx[TERR]+CREG: 0,1
Rx[TERR]OK
SYS_setTime: Adjusted system time by 166494 Secs (Src GPS Sync 1)
Read 248 bytes from CRUMBS.DAT

At 974679046 JS: lat = 32.832220, lon = -117.117098
At 975795232 JS: lat = 32.832222, lon = -117.117132
At 975795353 JS: lat = 32.832158, lon = -117.117085

Tx[TERR]AT+CSQ
Rx[TERR]+CSQ: 28,0
Rx[TERR]OK
  
```

**Figure 12-53: DemoAppFFS - Logger output for next Power On**

To remove the FFS file and start over, on the Logger port type 'd', 'f', 'r', 'CRUMBS.DAT.' This will remove the FFS file created by the application. The next time the application is run, a new file will be created.

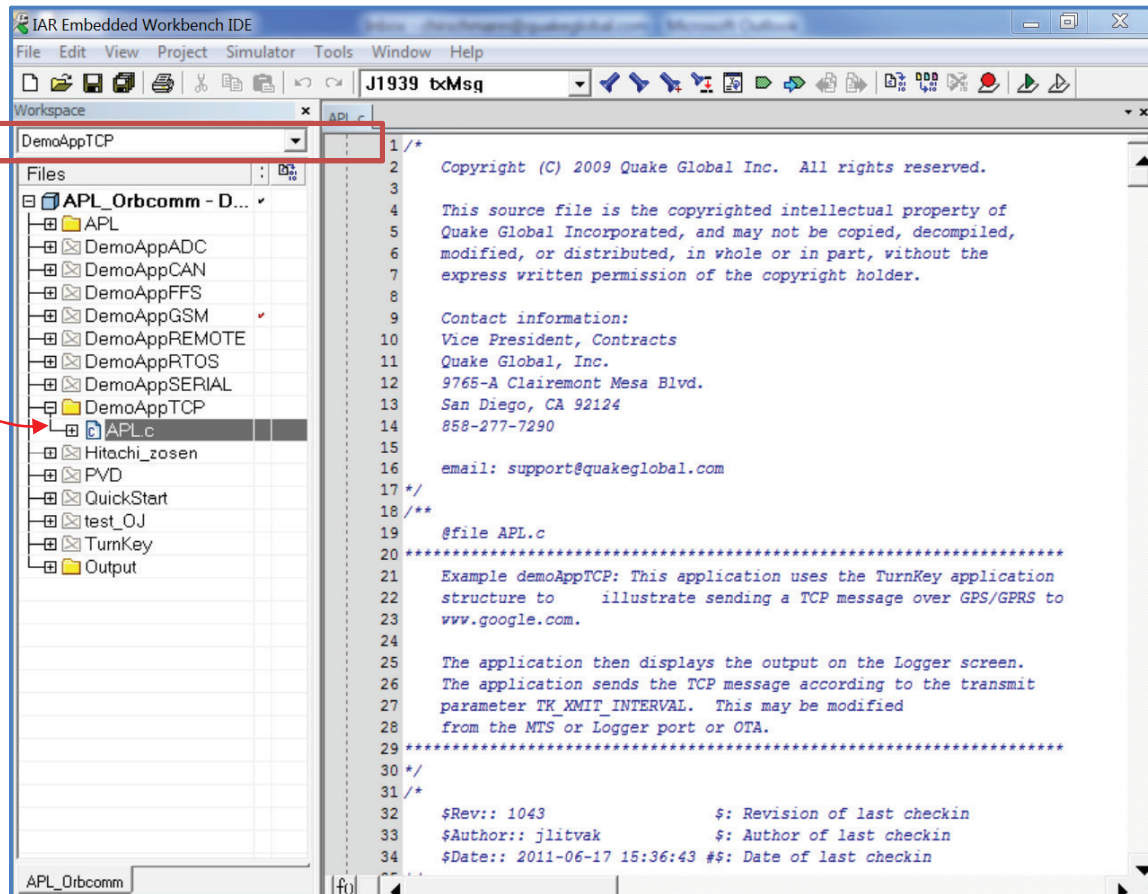
CONFIDENTIAL

Information classified Confidential - Do not copy (See last page for obligations)

## 12.4.6 DemoAppTCP

The DemoAppTCP sample application demonstrates using TCP to send and receive GSM/GPRS packets. It uses the Turnkey application structure to illustrate sending a TCP message over GPS/GPRS to [www.google.com](http://www.google.com). DemoAppTCP then displays the output on the Logger screen. Note that this sample application uses network-specific calls.

1. Select the DemoAppTCP Workspace from the drop-down list at the top, left-hand corner of the IAR IDE screen. Open the APL.c file, as shown below:

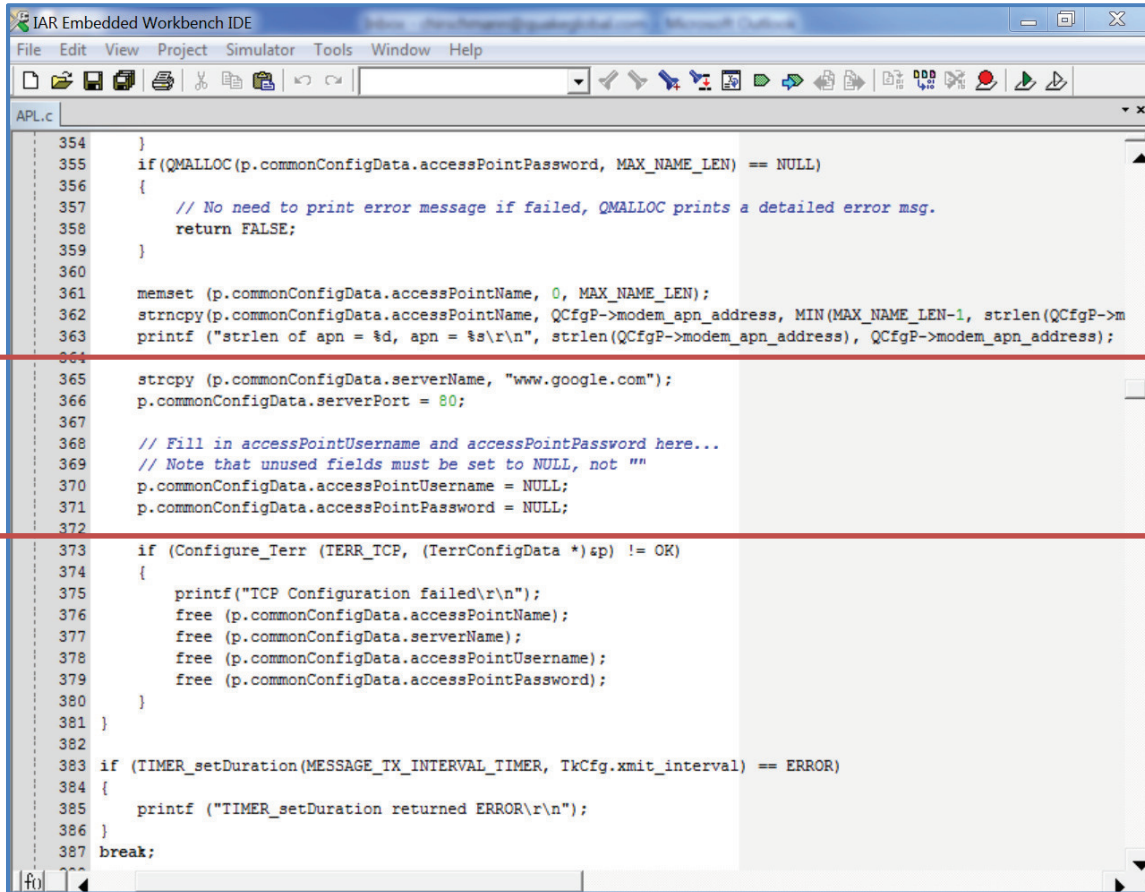


**Figure 12-54: DemoAppTCP - Selecting the Workspace**

2. Now build, load and execute DemoAppTCP. The instructions for building, loading and executing the code are the same as in [Section 12](#), except that after building the application, the executable bin file is: .../DemoAppTCP/exe/xxx-DemoAppTCP.bin.
3. After startup, check the Logger output for the line **APL DEMO: TCP**. This indicates that the correct DemoApp is running.

DemoAppTCP is based on the Turnkey sample application. The parameter `TK_XMIT_INTERVAL` is used to control how often a web page is requested. This may be modified from the MTS or Logger port or OTA. The web page is then displayed on the Logger output as text data.

The initialization of TCP upon POWER\_ON is shown in the following screen. The user should fill in his/her assigned access point username and password. Note that a web page from [www.google.com](http://www.google.com) is being requested. A timer is also set to expire after TK\_XMIT\_INTERVAL.



```

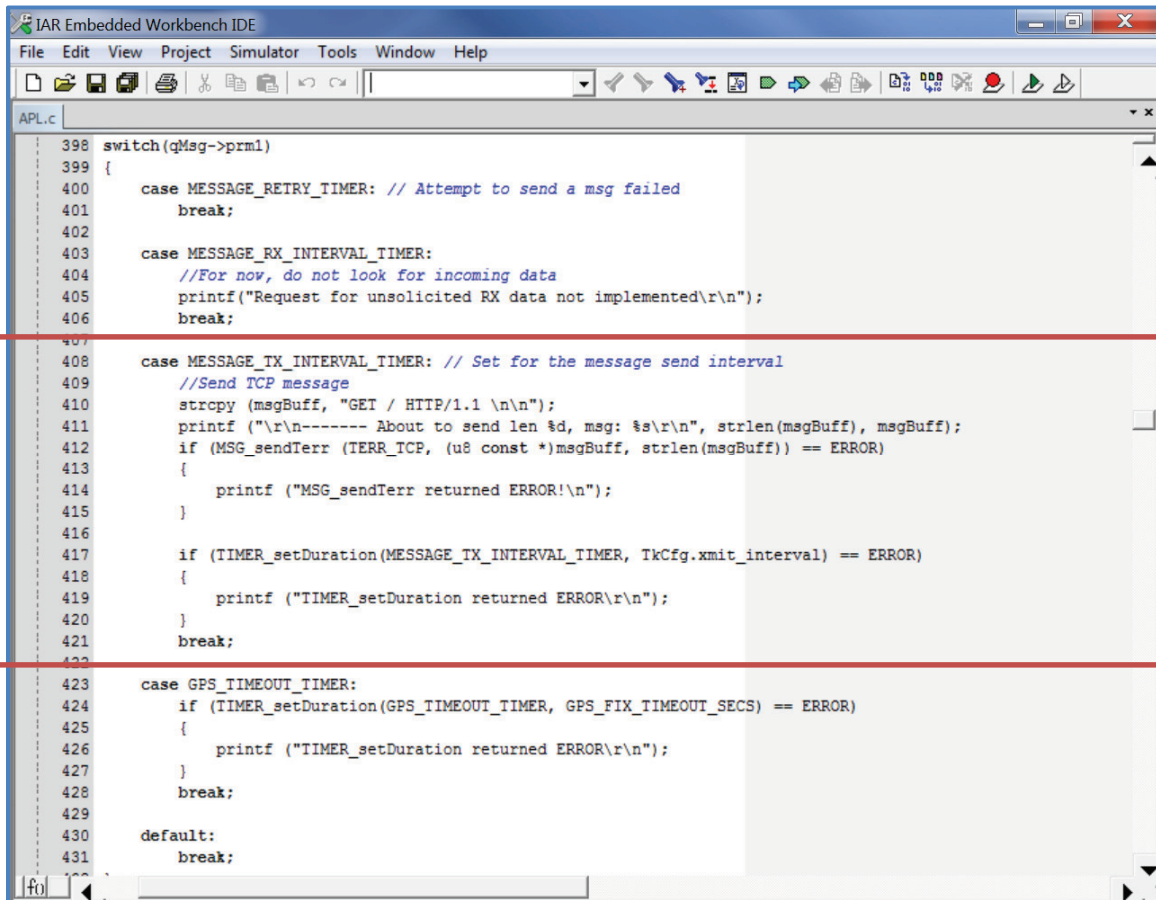
354 }
355 if(QMALLOC(p.commonConfigData.accessPointPassword, MAX_NAME_LEN) == NULL)
356 {
357     // No need to print error message if failed, QMALLOC prints a detailed error msg.
358     return FALSE;
359 }
360
361 memset (p.commonConfigData.accessPointName, 0, MAX_NAME_LEN);
362 strncpy(p.commonConfigData.accessPointName, QCfgP->modem_apn_address, MIN(MAX_NAME_LEN-1, strlen(QCfgP->m
363 printf ("strlen of apn = %d, apn = %s\r\n", strlen(QCfgP->modem_apn_address), QCfgP->modem_apn_address);
364
365 strcpy (p.commonConfigData.serverName, "www.google.com");
366 p.commonConfigData.serverPort = 80;
367
368 // Fill in accessPointUsername and accessPointPassword here...
369 // Note that unused fields must be set to NULL, not ""
370 p.commonConfigData.accessPointUsername = NULL;
371 p.commonConfigData.accessPointPassword = NULL;
372
373 if (Configure_Terr (TERR_TCP, (TerrConfigData *)&p) != OK)
374 {
375     printf("TCP Configuration failed\r\n");
376     free (p.commonConfigData.accessPointName);
377     free (p.commonConfigData.serverName);
378     free (p.commonConfigData.accessPointUsername);
379     free (p.commonConfigData.accessPointPassword);
380 }
381 }
382
383 if (TIMER_setDuration(MESSAGE_TX_INTERVAL_TIMER, TkCfg.xmit_interval) == ERROR)
384 {
385     printf ("TIMER_setDuration returned ERROR\r\n");
386 }
387 break;
    
```

Figure 12-55: DemoAppTCP - Initialization of TCP

CONFIDENTIAL

Information classified Confidential - Do not copy (See last page for obligations)

After the timer expires, MSG\_sendTerr() is called to receive the web page data. The timer is then reset to expire after TK\_XMIT\_INTERVAL.



```

398 switch(qMsg->prml)
399 {
400     case MESSAGE_RETRY_TIMER: // Attempt to send a msg failed
401         break;
402     case MESSAGE_RX_INTERVAL_TIMER:
403         //For now, do not look for incoming data
404         printf("Request for unsolicited RX data not implemented\r\n");
405         break;
406     case MESSAGE_TX_INTERVAL_TIMER: // Set for the message send interval
407         //Send TCP message
408         strcpy(msgBuff, "GET / HTTP/1.1 \n\n");
409         printf("\r\n----- About to send len %d, msg: %s\r\n", strlen(msgBuff), msgBuff);
410         if (MSG_sendTerr(TERR_TCP, (u8 const *)msgBuff, strlen(msgBuff)) == ERROR)
411         {
412             printf("MSG_sendTerr returned ERROR!\n");
413         }
414         if (TIMER_setDuration(MESSAGE_TX_INTERVAL_TIMER, TkCfg.xmit_interval) == ERROR)
415         {
416             printf("TIMER_setDuration returned ERROR\r\n");
417         }
418         break;
419     case GPS_TIMEOUT_TIMER:
420         if (TIMER_setDuration(GPS_TIMEOUT_TIMER, GPS_FIX_TIMEOUT_SECS) == ERROR)
421         {
422             printf("TIMER_setDuration returned ERROR\r\n");
423         }
424         break;
425     default:
426         break;
427 }

```

Figure 12-56: DemoAppTCP - Request for web page data

CONFIDENTIAL

Information classified Confidential - Do not copy (See last page for obligations)



The application is notified that a message has been received by the MSG\_RCVD event. Here the web data are printed out to the Logger port, as shown below.

```

LOGGER - HyperTerminal
File Edit View Call Transfer Help

Rx[TERR]OK
RECEIVED MSG = 1e.jsrt_kill=1;
                                var_gjwl=location;function_gjuc(){var_e=gjwl.hr
ef.indexOf("#");if(e>=0){var_a=gjwl.href.substring(e);if(a.indexOf("&q=")>0){a.
indexOf("#q=")>0){a=a.substring(1);if(a.indexOf("#")==-1){for(var_c=0;c<a.lengt
h;){var_d=c;ifTx[TERR]AT#SI=2
Rx[TERR]#SI: 2,17,2760,3330,0
Rx[TERR]OK
Tx[TERR]AT#SRECV=2,1500
Rx[TERR]#SRECV: 2,1500
Rx[TERR]OK
RECEIVED MSG = gbs,.gbm| background:#fff;left:0;position:absolute;text-align:left
;visiblity:hidden;z-index:1000|.gbm| border:1px solid;border-color:#c9d7f1 #36c
#36c #a2bae7;z-index:1001|.gb1| margin-right:.5em|#gbar .gbsup| color:#c00;font-si
ze:9px;font-weitx[TERR]AT#SRECV=2,1500
Rx[TERR]#SRECV: 2,1500
Rx[TERR]OK
RECEIVED MSG = splay:block!important|</style><script>google.y=| |;google.x=functi
on(e,g)| google.y[e.id]=[e,g];return false|;window.gbar=| qs:function(){| ,tg:funct
ion(e)| var_o=id:'gbar'|;for(i in e)o[i]=e[i];google.x(o,function())| gbar.tg(o) |)
| |;</script></hTx[TERR]AT#SRECV=2,330
Rx[TERR]#SRECV: 2,330
Rx[TERR]OK
RECEIVED MSG = href="http://translate.google.com/?hl=en&tab=wT" onclick=gbar.qs
(this) class=gb2>Translate</a> <a href="http://scholar.google.com/schhp?hl=en&ta
b=ws" onclick=gbar.qs(this) class=gb2>Scholar</a> <a href="http://blogsearch.goo
gle.com/?hl=en&Tx[TERR]AT#SI=2
Rx[TERR]#SI: 2,17,6090,5394,0
Rx[TERR]OK
Tx[TERR]AT#SRECV=2,1500
Rx[TERR]#SRECV: 2,1500
Rx[TERR]OK
RECEIVED MSG = hl=en&tab=wY" onclick=gbar.qs(this) class=gb2>Updates</a> <div c1
ass=gb2><div class=gb2></div></div><a href="http://www.youtube.com/?hl=en&tab=w1
" onclick=gbar.qs(this) class=gb2>YouTube</a> <a href="http://www.google.com/cal
endar/render?hlTx[TERR]AT#SRECV=2,1500
Rx[TERR]#SRECV: 2,1500
Rx[TERR]OK
RECEIVED MSG = 0 14px" onload="window.lol&lol()"><br><br></div><form action="/"
search" name=f><table cellpadding=0 cellspacing=0><tr valign=top><td width=25%>&
nbsp;</td><td align=center nowrap><input name=hl type=hidden value=en><input nam
e=source type=hTx[TERR]AT#SRECV=2,1500
Rx[TERR]#SRECV: 2,1500
Rx[TERR]OK
RECEIVED MSG = tl/en/privacy.html">Privacy</a></p></center></span> <div id=xjsd>
</div><div id=xjsi><script>if(google.y)google.y.first=[];google.dlj=function(b)|
window.setTimeout(function(){var_a=document.createElement("script");a.src=b;docu
ment.getElementTx[TERR]AT#SRECV=2,894
Rx[TERR]#SRECV: 2,894

```

Connected 0:17:00 VT100 115200 8-N-1 SCROLL CAPS NUM Capture Print echo

Figure 12-57: DemoAppTCP - Web data displayed on the Logger port

CONFIDENTIAL

Information classified Confidential - Do not copy (See last page for obligations)

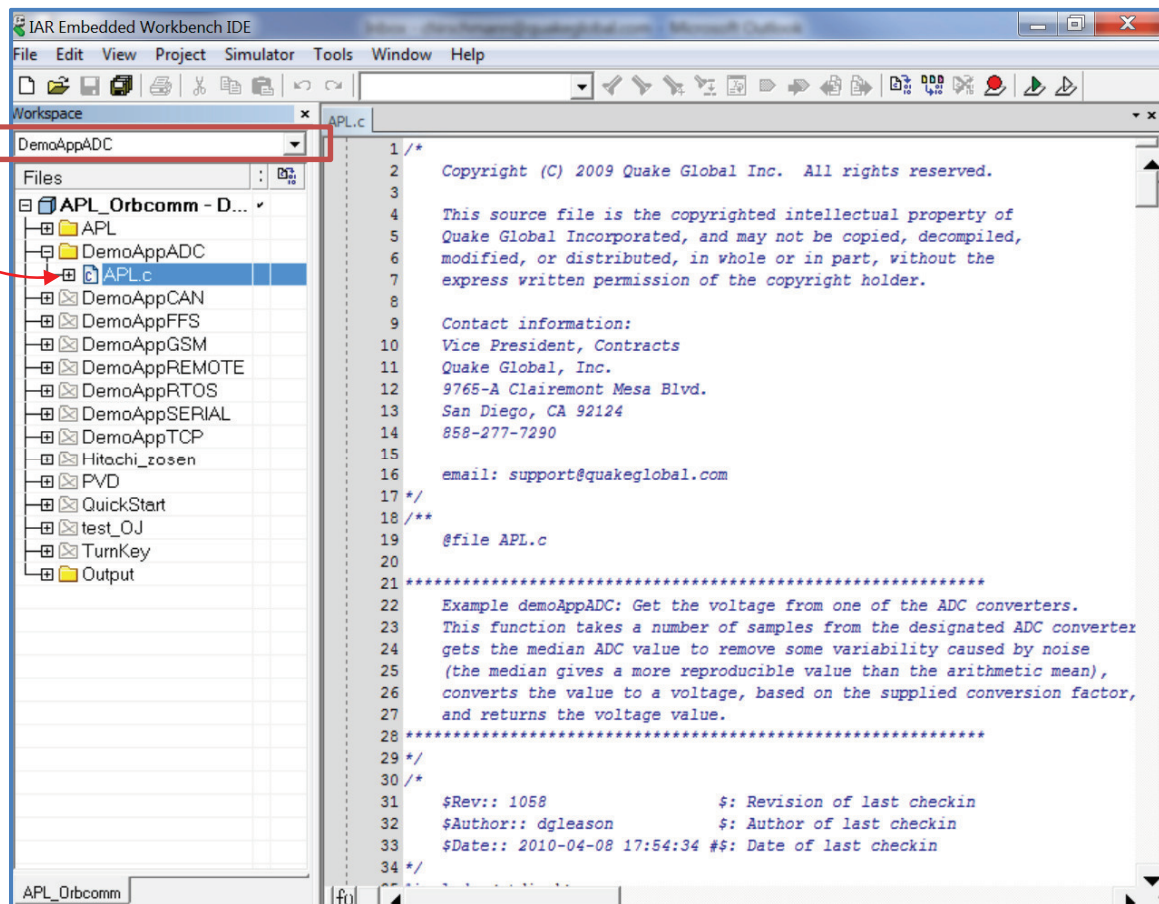
## 12.4.7 DemoAppADC

The Analog to Digital Converter (ADC) example:

- takes a number of samples from the designated ADC
- gets the median ADC value to remove some variability caused by noise (the median gives a more reproducible value than the arithmetic mean)
- converts the value to a voltage based on the supplied conversion factor
- prints the voltage value to the Logger port.

This process repeats approximately once a second until the demo is halted. Note that this application uses network-specific calls.

1. Select the DemoAppADC Workspace from the drop-down list at the top left-hand corner of the IAR IDE screen. Open the APL.c file, as shown below:



**Figure 12-58: DemoAppADC - Selecting the Workspace**

2. Now build, load and execute DemoAppADC. The instructions for building, loading and executing the code are the same as in [Section 12](#), except that after building the application, the executable bin file is: .../DemoAppADC/exe/xxx-DemoAppADC.bin.

- After startup, check the Logger output for the line **APL DEMO: ADC**. This indicates that the correct DemoApp is running.

The ADC converter is selected by #define statements in the code, as shown in Figure 12-59.

```

139 */
140
141 #define ADC_READ_INPUT_VOLTAGE_CHAN 2
142 #define MAX_NUM_VOLT_READINGS 401
143 #define VOLTAGE_ERROR -1.0
144 #define DEFAULT_ADC_SAMPLE_CNT 51
145 #define DEBUG_ADC_GETVOLTAGE
146
147 /**
148  Get the voltage from one of the ADC converters. This function takes
149  a number of samples from the designated ADC converter, gets the median
150  ADC value to remove some variability caused by noise (the median gives
151  a more reproducible value than the arithmetic mean), converts the value
152  to a voltage, based on the supplied conversion factor, and returns the
153  voltage value.
154
155  @note Undefine DEBUG_ADC_GETVOLTAGE to remove some debug prints
156
157  @param adcNum Which ADC to use for the reading (ADC #2 is monitors supply voltage)
158  @param numSamp how many samples to take; must not exceed MAX_NUM_VOLT_READINGS
159  @param conversionFaction Multiplier used to convert ADC counts to volts
160
161  @return The voltage on the ADC's input, or VOLTAGE_ERROR on error
162 */

```

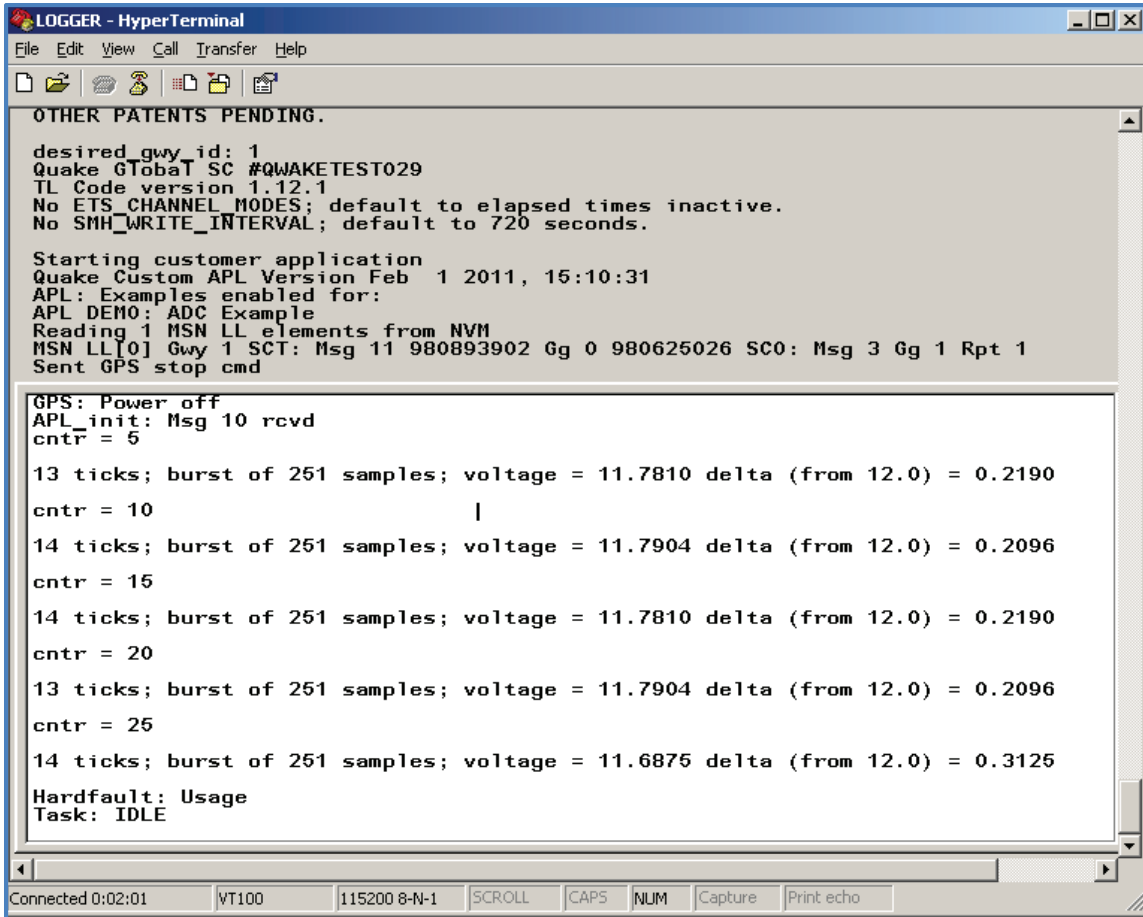
Figure 12-59: DemoAppADC - Definitions

CONFIDENTIAL

Information classified Confidential - Do not copy (See last page for obligations)



The application prints the ADC data approximately once per second, as shown in Figure 12-60.



```

LOGGER - HyperTerminal
File Edit View Call Transfer Help

OTHER PATENTS PENDING.

desired_gwy_id: 1
Quake GTobaT SC #QWAKETEST029
TL Code version 1.12.1
No ETS_CHANNEL_MODES; default to elapsed times inactive.
No SMH_WRITE_INTERVAL; default to 720 seconds.

Starting customer application
Quake Custom APL Version Feb 1 2011, 15:10:31
APL: Examples enabled for:
APL DEMO: ADC Example
Reading 1 MSN LL elements from NVM
MSN LL[0] Gwy 1 SCT: Msg 11 980893902 Gg 0 980625026 SC0: Msg 3 Gg 1 Rpt 1
Sent GPS stop cmd

GPS: Power off
APL_init: Msg 10 rcvd
cntr = 5

13 ticks; burst of 251 samples; voltage = 11.7810 delta (from 12.0) = 0.2190
cntr = 10
14 ticks; burst of 251 samples; voltage = 11.7904 delta (from 12.0) = 0.2096
cntr = 15
14 ticks; burst of 251 samples; voltage = 11.7810 delta (from 12.0) = 0.2190
cntr = 20
13 ticks; burst of 251 samples; voltage = 11.7904 delta (from 12.0) = 0.2096
cntr = 25
14 ticks; burst of 251 samples; voltage = 11.6875 delta (from 12.0) = 0.3125

Hardfault: Usage
Task: IDLE

Connected 0:02:01 VT100 115200 8-N-1 SCROLL CAPS NUM Capture Print echo
  
```

Figure 12-60: DemoAppADC - Logger data

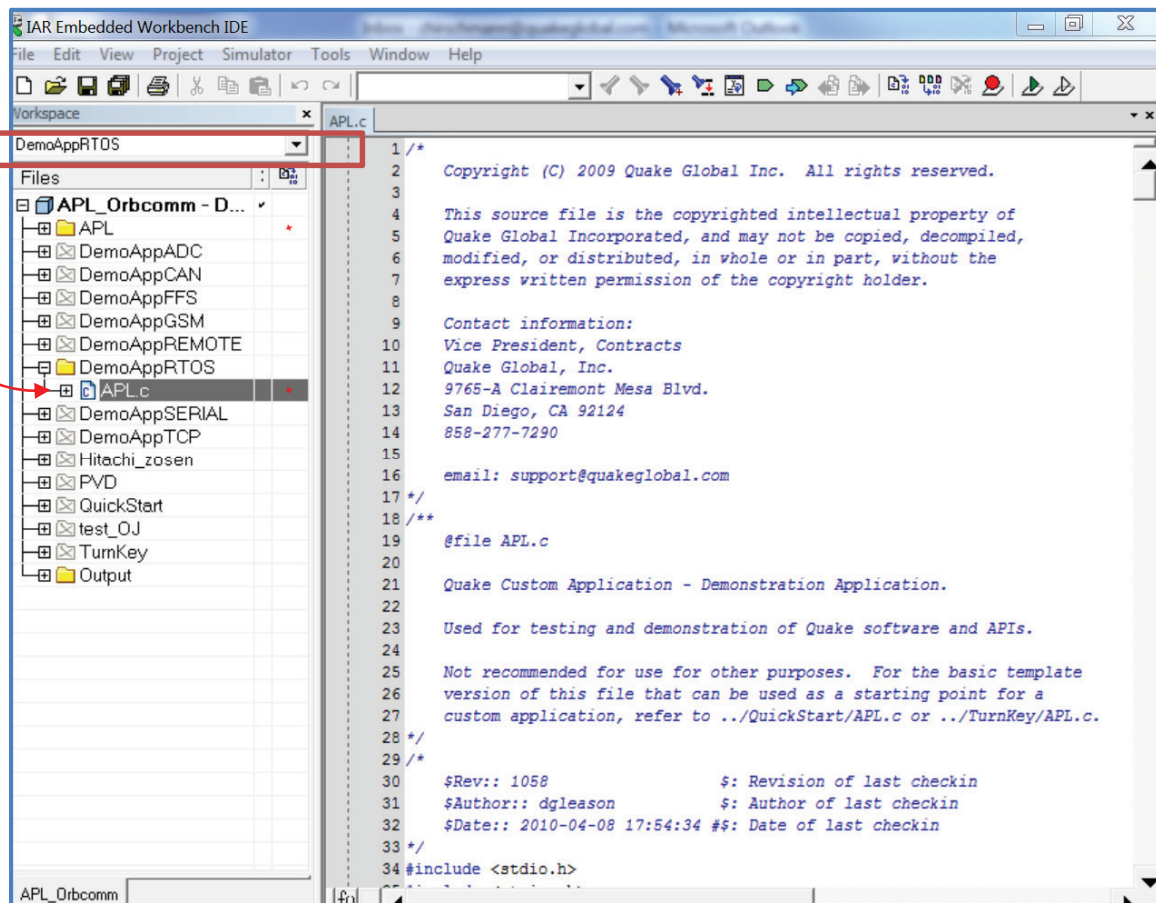
CONFIDENTIAL

Information classified Confidential - Do not copy (See last page for obligations)

## 12.4.8 DemoAppRTOS

The Real-Time Operating System (DemoAppRTOS) sample application demonstrates use of various RTOS and related system features. The primary features demonstrated are creation of two RTOS tasks which run independently, and creation and use of a task message queue to communicate between the tasks. However, other features are also exercised, including the use of a mutex semaphore to protect a shared resource, and access to the system tick counter and error reporting facility. Note that this application uses network-specific calls.

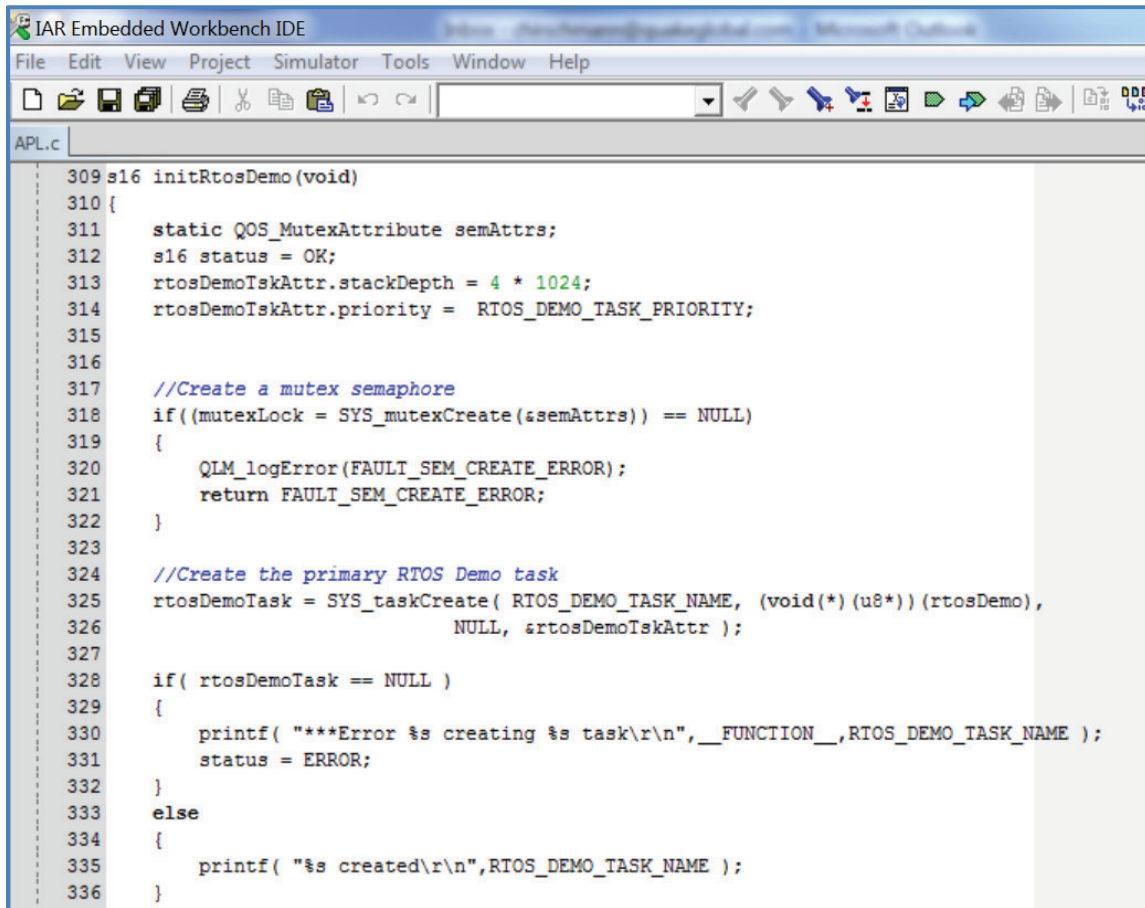
1. Select the DemoAppRTOS Workspace from the drop-down list at the top left-hand corner of the IAR IDE screen. Open the APL.c file, as shown below:



**Figure 12-61: DemoAppRTOS - Selecting the Workspace**

2. Now build, load and execute DemoAppRTOS. The instructions for building, loading and executing the code are the same as in [Section 12](#), except that after building the application, the executable bin file is: .../DemoAppRTOS/exe/xxx-DemoAppRTOS.bin.
3. After startup, check the Logger output for the line **APL DEMO: RTOS**. This indicates that the correct DemoApp is running.

The `initRtosDemo()` function first creates a mutex semaphore and then creates the primary RTOS Demo task.



```

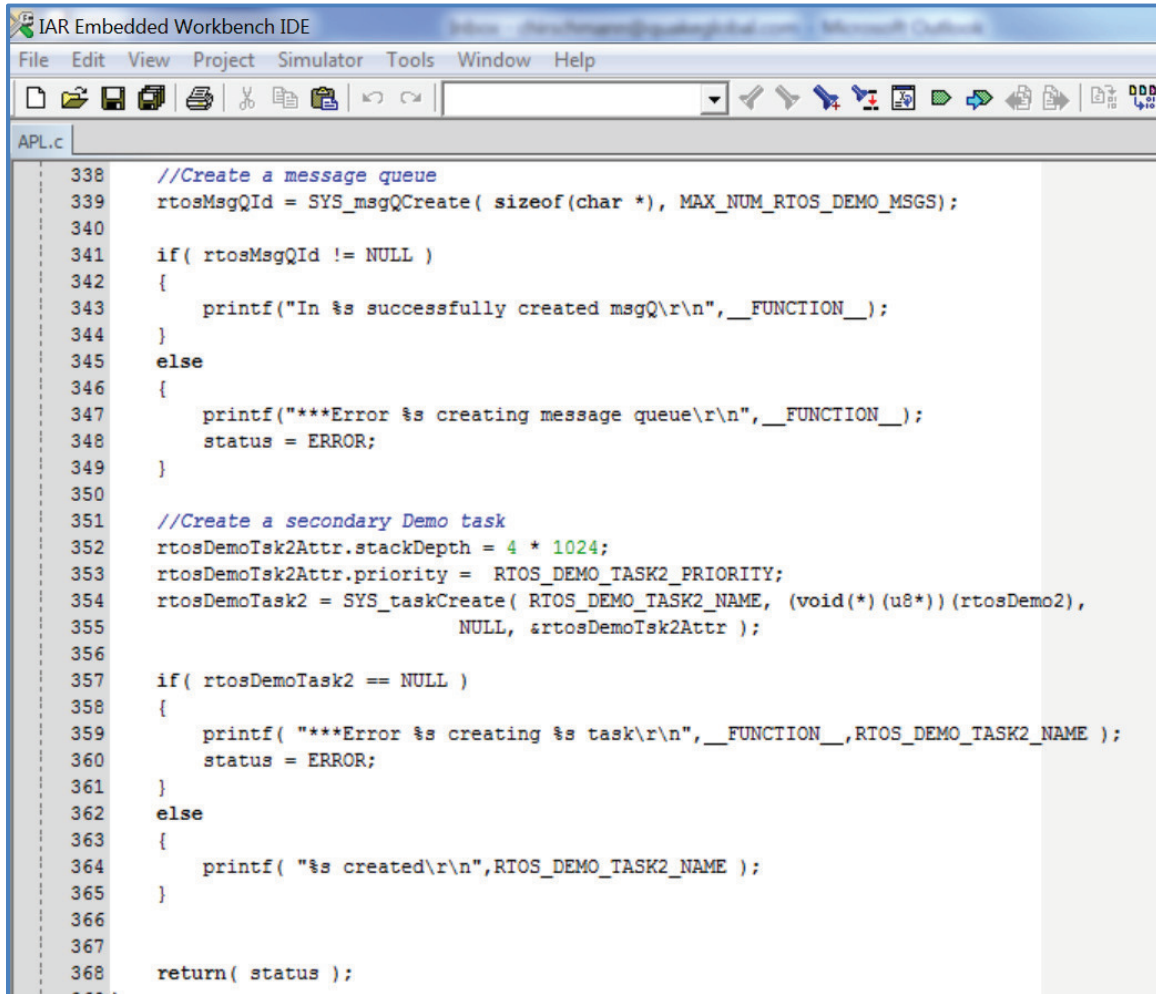
309 s16 initRtosDemo(void)
310 {
311     static QOS_MutexAttribute semAttrs;
312     s16 status = OK;
313     rtosDemoTaskAttr.stackDepth = 4 * 1024;
314     rtosDemoTaskAttr.priority = RTOS_DEMO_TASK_PRIORITY;
315
316     //Create a mutex semaphore
317     if((mutexLock = SYS_mutexCreate(&semAttrs)) == NULL)
318     {
319         QLM_logError(Fault_Sem_Create_Error);
320         return Fault_Sem_Create_Error;
321     }
322
323     //Create the primary RTOS Demo task
324     rtosDemoTask = SYS_taskCreate( RTOS_DEMO_TASK_NAME, (void(*) (u8*)) (rtosDemo),
325                                   NULL, &rtosDemoTaskAttr );
326
327     if( rtosDemoTask == NULL )
328     {
329         printf( "***Error %s creating %s task\r\n", __FUNCTION__, RTOS_DEMO_TASK_NAME );
330         status = ERROR;
331     }
332     else
333     {
334         printf( "%s created\r\n", RTOS_DEMO_TASK_NAME );
335     }
336 }
  
```

Figure 12-62: DemoAppRTOS - initDemoRTOS (view 1)

CONFIDENTIAL

Information classified Confidential - Do not copy (See last page for obligations)

The `initRtosDemo()` function then creates a message queue and a secondary Demo task.



```

338 //Create a message queue
339 rtosMsgQId = SYS_msgQCreate( sizeof(char *), MAX_NUM_RTOS_DEMO_MSGS);
340
341 if( rtosMsgQId != NULL )
342 {
343     printf("In %s successfully created msgQ\r\n",__FUNCTION__);
344 }
345 else
346 {
347     printf("****Error %s creating message queue\r\n",__FUNCTION__);
348     status = ERROR;
349 }
350
351 //Create a secondary Demo task
352 rtosDemoTask2Attr.stackDepth = 4 * 1024;
353 rtosDemoTask2Attr.priority = RTOS_DEMO_TASK2_PRIORITY;
354 rtosDemoTask2 = SYS_taskCreate( RTOS_DEMO_TASK2_NAME, (void(*) (u8*)) (rtosDemo2),
355                                NULL, &rtosDemoTask2Attr );
356
357 if( rtosDemoTask2 == NULL )
358 {
359     printf( "****Error %s creating %s task\r\n",__FUNCTION__,RTOS_DEMO_TASK2_NAME );
360     status = ERROR;
361 }
362 else
363 {
364     printf( "%s created\r\n",RTOS_DEMO_TASK2_NAME );
365 }
366
367
368 return( status );

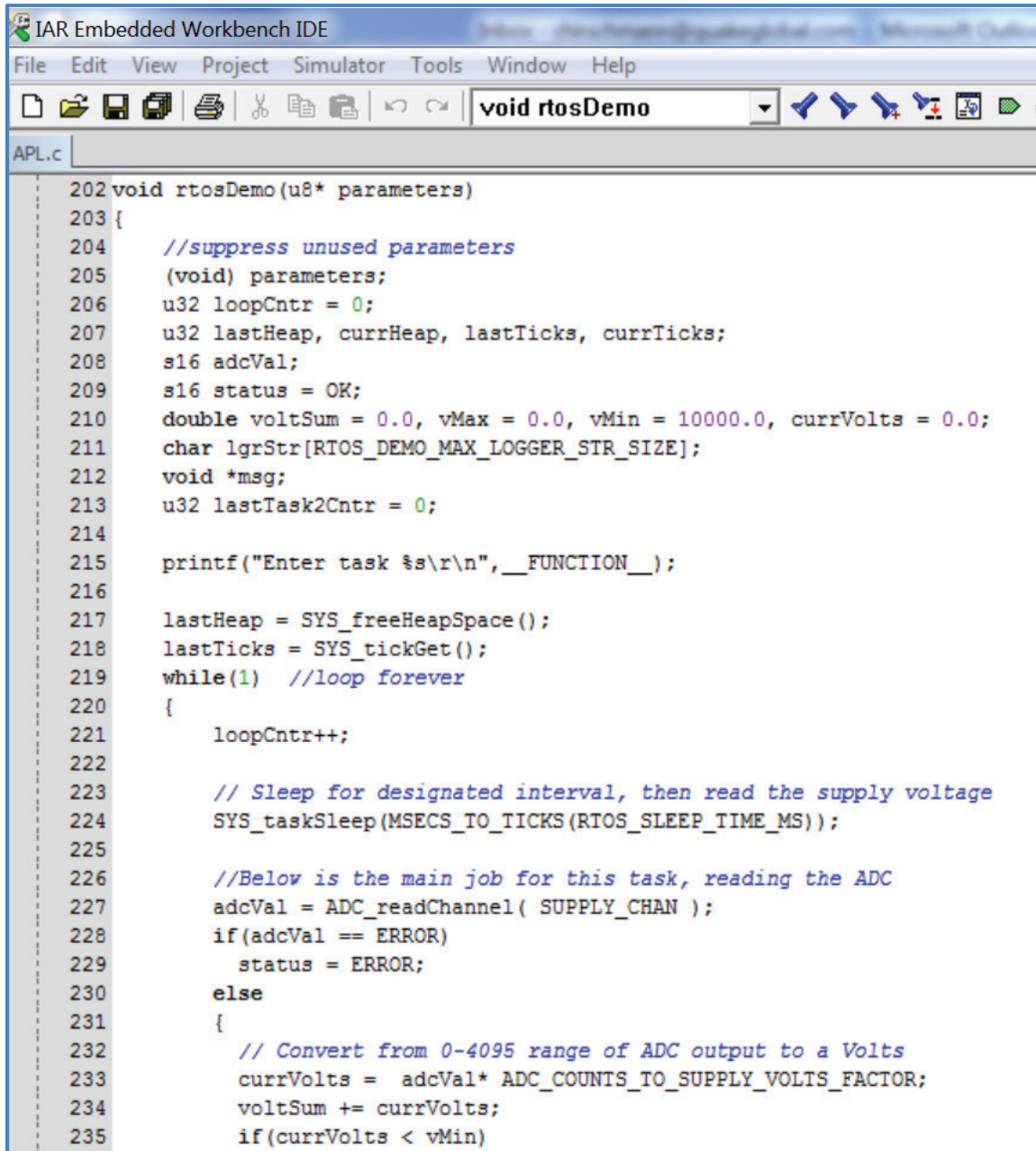
```

Figure 12-63: DemoAppRTOS - initDemoRTOS (view 2)

CONFIDENTIAL

Information classified Confidential - Do not copy (See last page for obligations)

The first demo task to be created, `rtosDemo()`, is the primary task. It samples the supply/battery voltage and does some rudimentary analysis of the data. The task also periodically creates a formatted printable string which it sends to the secondary task via a message queue to be printed. It then demonstrates the use of a mutex and periodically checks the heap to demonstrate use of this feature.



```

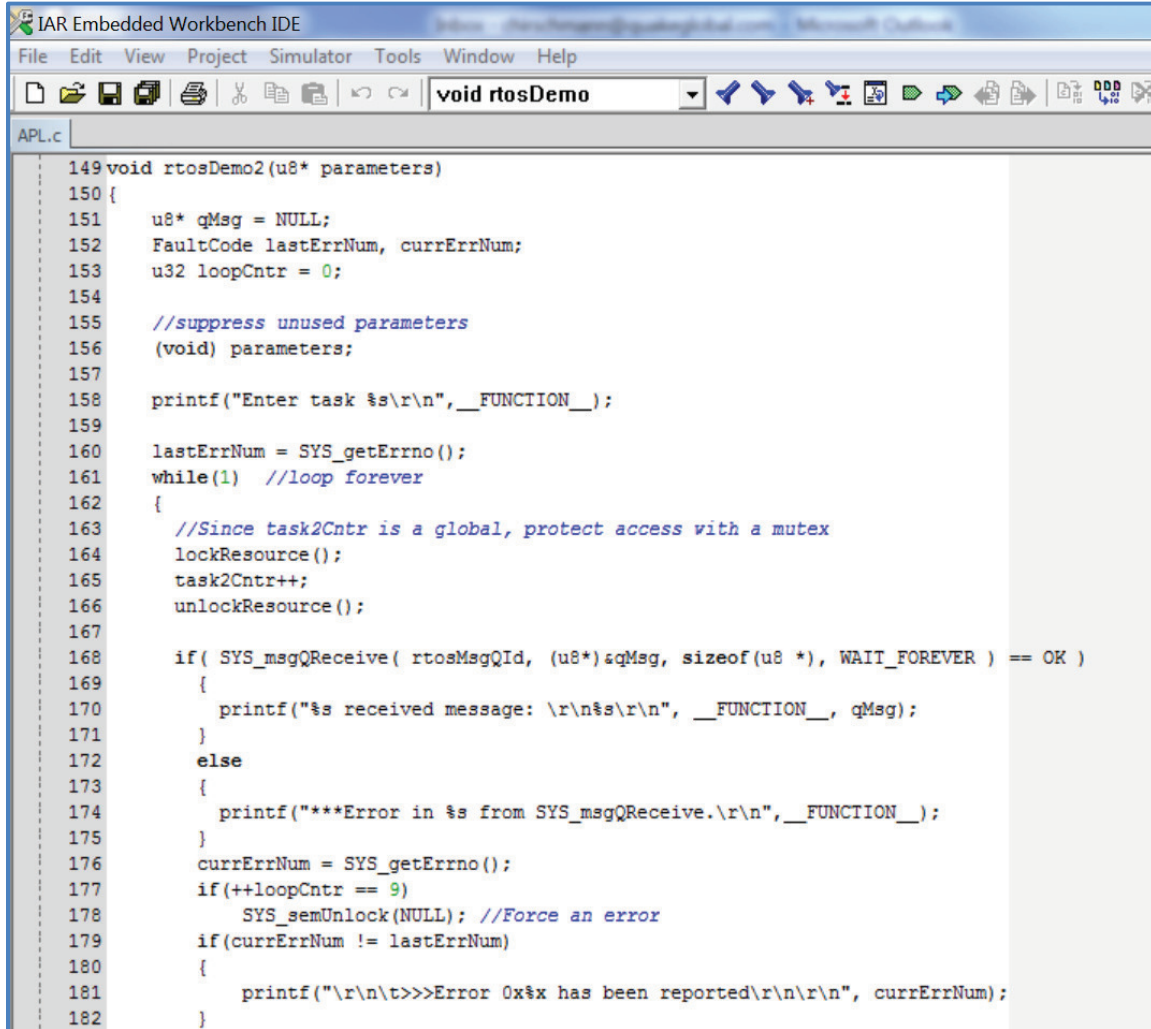
202 void rtosDemo(u8* parameters)
203 {
204     //suppress unused parameters
205     (void) parameters;
206     u32 loopCntr = 0;
207     u32 lastHeap, currHeap, lastTicks, currTicks;
208     s16 adcVal;
209     s16 status = OK;
210     double voltSum = 0.0, vMax = 0.0, vMin = 10000.0, currVolts = 0.0;
211     char lgrStr[RTOS_DEMO_MAX_LOGGER_STR_SIZE];
212     void *msg;
213     u32 lastTask2Cntr = 0;
214
215     printf("Enter task %s\r\n", __FUNCTION__);
216
217     lastHeap = SYS_freeHeapSpace();
218     lastTicks = SYS_tickGet();
219     while(1) //loop forever
220     {
221         loopCntr++;
222
223         // Sleep for designated interval, then read the supply voltage
224         SYS_taskSleep(MSECS_TO_TICKS(RTOS_SLEEP_TIME_MS));
225
226         //Below is the main job for this task, reading the ADC
227         adcVal = ADC_readChannel( SUPPLY_CHAN );
228         if(adcVal == ERROR)
229             status = ERROR;
230         else
231         {
232             // Convert from 0-4095 range of ADC output to a Volts
233             currVolts = adcVal* ADC_COUNTS_TO_SUPPLY_VOLTS_FACTOR;
234             voltSum += currVolts;
235             if(currVolts < vMin)

```

Figure 12-64: DemoAppRTOS - Primary Demo task



The RTOS demo secondary task, `rtosDemo2()`, receives a message from the initial RTOS task. The message contains some statistics that the main task has gathered and formatted into a printable string. The secondary task then prints the string.



```

149 void rtosDemo2(u8* parameters)
150 {
151     u8* qMsg = NULL;
152     FaultCode lastErrNum, currErrNum;
153     u32 loopCntr = 0;
154
155     //suppress unused parameters
156     (void) parameters;
157
158     printf("Enter task %s\r\n", __FUNCTION__);
159
160     lastErrNum = SYS_getErrno();
161     while(1) //loop forever
162     {
163         //Since task2Cntr is a global, protect access with a mutex
164         lockResource();
165         task2Cntr++;
166         unlockResource();
167
168         if( SYS_msgQReceive( rtosMsgQId, (u8*)&qMsg, sizeof(u8 *), WAIT_FOREVER ) == OK )
169         {
170             printf("%s received message: \r\n%s\r\n", __FUNCTION__, qMsg);
171         }
172         else
173         {
174             printf("****Error in %s from SYS_msgQReceive.\r\n", __FUNCTION__);
175         }
176         currErrNum = SYS_getErrno();
177         if(++loopCntr == 9)
178             SYS_semUnlock(NULL); //Force an error
179         if(currErrNum != lastErrNum)
180         {
181             printf("\r\n\t>>>Error 0x%x has been reported\r\n\r\n", currErrNum);
182         }
183     }

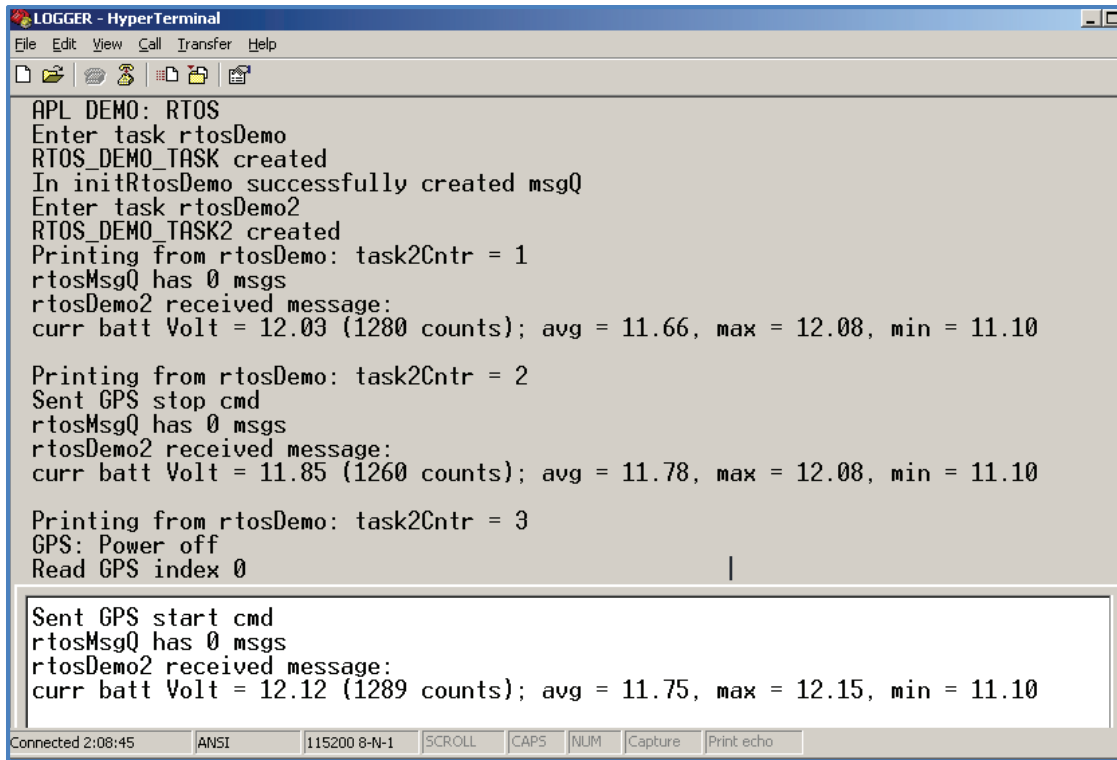
```

**Figure 12-65: DemoAppRTOS - Secondary Demo task**

CONFIDENTIAL

Information classified Confidential - Do not copy (See last page for obligations)

The Logger output from DemoAppRTOS is displayed in Figure 12-66.



```

LOGGER - HyperTerminal
File Edit View Call Transfer Help
APL DEMO: RTOS
Enter task rtosDemo
RTOS_DEMO_TASK created
In initRtosDemo successfully created msgQ
Enter task rtosDemo2
RTOS_DEMO_TASK2 created
Printing from rtosDemo: task2Cntr = 1
rtosMsgQ has 0 msgs
rtosDemo2 received message:
curr batt Volt = 12.03 (1280 counts); avg = 11.66, max = 12.08, min = 11.10

Printing from rtosDemo: task2Cntr = 2
Sent GPS stop cmd
rtosMsgQ has 0 msgs
rtosDemo2 received message:
curr batt Volt = 11.85 (1260 counts); avg = 11.78, max = 12.08, min = 11.10

Printing from rtosDemo: task2Cntr = 3
GPS: Power off
Read GPS index 0

Sent GPS start cmd
rtosMsgQ has 0 msgs
rtosDemo2 received message:
curr batt Volt = 12.12 (1289 counts); avg = 11.75, max = 12.15, min = 11.10
    
```

**Figure 12-66: DemoAppRTOS - Logger output**

CONFIDENTIAL

Information classified Confidential - Do not copy (See last page for obligations)

## 13 Satellite networks

### 13.1 ORBCOMM network

The ORBCOMM System is a wide area, packet-switched, two-way data communication system. Communications between modems and ORBCOMM Gateways are accomplished via a constellation of Low-Earth Orbit (LEO) satellites.

The ORBCOMM System consists of ground infrastructure, a space segment, and a subscriber segment. A Network Control Center (NCC) manages the overall system worldwide.

- The **ground infrastructure** contains the ORBCOMM Gateway, which provides message processing and subscriber management.
- The **space segment** currently consists of 29 LEO satellites and one Satellite Control Center (SCC).
- The **subscriber segment** consists of the modems used by ORBCOMM System subscribers to transmit information to, and receive information from the LEO satellites.

RF communication within the ORBCOMM System operates in the Very High Frequency (VHF) portion of the frequency spectrum, between 137 and 150 Megahertz (MHz).

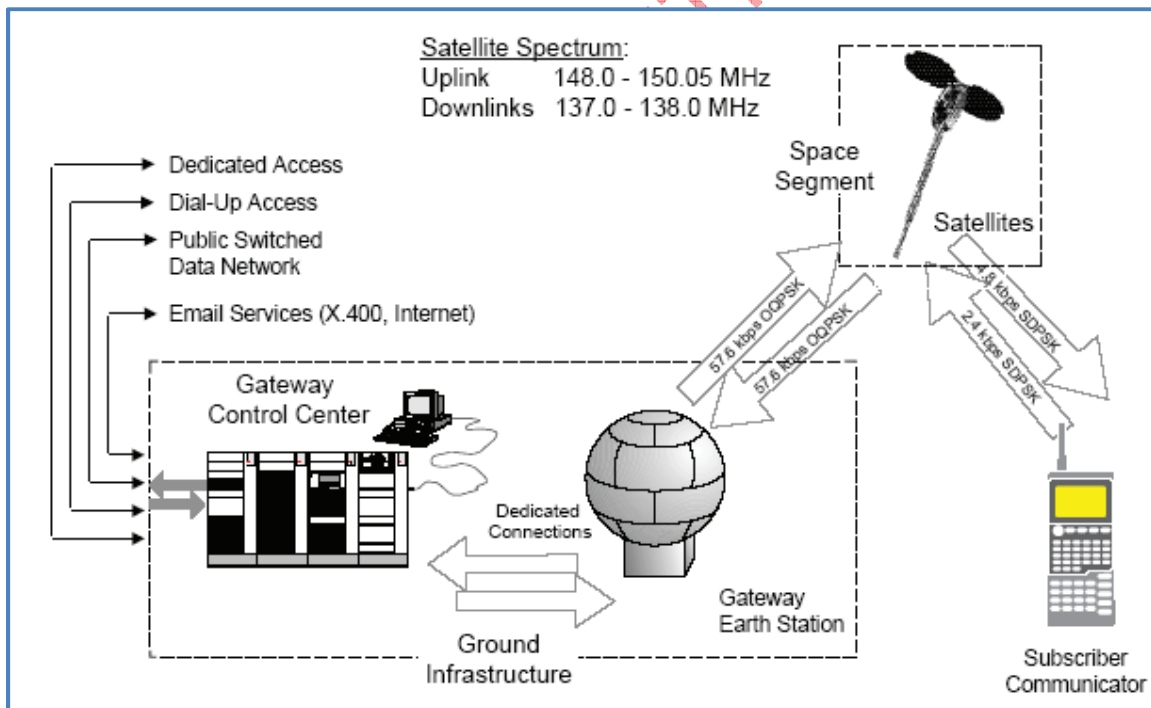


Figure 13-1: ORBCOMM satellite network diagram

There are four basic service elements provided by the ORBCOMM System: Data Reports, Messages, Globalgrams, and Commands.

- **Data Reports** – the modem uses a data report to transmit or receive a single packet containing 6 bytes or less of user-defined data.

- **Messages** – the modem uses a message to transmit or receive a longer sequence of data. Messages typically have lengths less than 100 bytes, although the ORBCOMM System can handle longer messages (8 Kbytes maximum, but messages of less than 1000 bytes are recommended).
- **Globalgrams** – the modem uses a Globalgram to transmit or receive a single, self-contained data packet to or from a satellite that is not in view of an ORBCOMM Gateway. A modem-originated Globalgram contains up to 229 bytes of user data.
- **User Commands** – a user command is used to transmit a single packet containing 5 bytes or less of user-defined data.

See [Appendix A](#) for a list of ORBCOMM Configuration Parameters.

### 13.1.1 ORBCOMM Auto-Roaming

A message can be automatically sent to any Gateway with which the modem is provisioned. This ensures that the application can communicate anywhere in the world with no extra programming required.

Auto-Roaming should be enabled in the QUAKE configuration parameter **QCFG MTS AUTO ROAMING ENA** if your application meets one or more of the following criteria:

- Your modem is likely to roam outside of your main Gateway service area.
- Your modem is provisioned with more than one Gateway, and you want your message(s) to be automatically routed through any Gateway with which the modem is provisioned.
- Your modem is likely to operate in an area where a Gateway is not always in view of the satellites, and you want to automatically send messages as either Globalgrams or Messages/Reports, depending on Gateway availability.

When a message is sent with Auto-Roaming enabled, the modem monitors the message and the status of the satellite(s) and Gateway(s) in view until the message is successfully delivered. If changes occur in the status of available satellites or Gateways, your message is automatically re-routed in accordance with the new communications conditions. This process continues indefinitely until the message is successfully delivered. This ensures that messages are delivered as quickly and efficiently as possible.

There are two classes of Auto-Roaming functionality: Globalgrams and all other message types. The Auto-Roaming option enables a Globalgram to be sent as any message type to any Gateway or satellite. If no Gateway is in view of the satellite when the message is formed, it is sent as a Globalgram. If a desired Gateway is in view, the Globalgram is automatically converted to a message or a report if the length of the message data is 6 bytes or less. Use of the Globalgram message type with Auto-Roaming enabled provides the fastest possible communications regardless of where the modem is located.

For message types other than Globalgrams, the Auto-Roaming functionality is similar, with the exception that these message types cannot be converted to Globalgrams. If a message or report is created with Auto-Roaming enabled, it is automatically sent to any desirable Gateway that is in view, and is automatically adjusted should the Gateway status change before the message is successfully delivered. These message types are not converted to Globalgrams if there are no Gateways in view. This is because high message latencies can occur with Globalgrams in certain areas, and this latency can be improved by waiting until the modem is in view of a desirable Gateway, rather than sending the message as a Globalgram.

For roaming applications that demand the absolute minimum message latency, it is suggested to send two messages: one as a Globalgram, and the other as a message or report. This ensures that both of these delivery avenues are utilized, but may result in higher airtime costs.



**Note:**

The term “desirable Gateway” refers to the Desired Gateways List that is maintained in the modem’s Non-Volatile Memory (NVM). If no list has been entered into the modem, all Gateways are considered desirable. If a Desired Gateways List has been entered, however, communications are only allowed (or not allowed, depending on how the list is configured) with those gateways specified in the list.

## 13.2 Iridium network

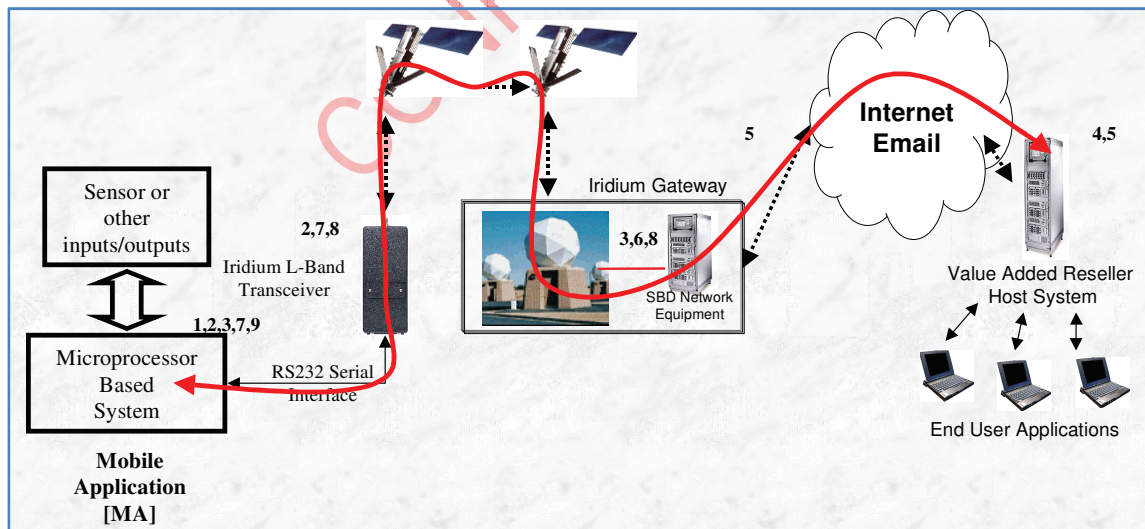
The Iridium Satellite Network is a world-wide, two-way data communication system. Communications between modems and the Iridium ground network are accomplished via a constellation of low-earth orbit (LEO) satellites. There are 66 operational satellites with additional spares that operate in 6 polar planes with pole to pole coverage at all times.

The ground network is comprised of the System Control Segment and gateways used to connect to the terrestrial data networks. The System Control Segment is the central management component for the Iridium Satellite Network. It provides support and control services for the satellite constellation and delivers satellite tracking data to the gateways.



**Note:**

The maximum mobile originated message size through the Iridium network is 340 bytes. The maximum mobile terminated message size is 270 bytes.



**Figure 13-2: Iridium network diagram**

CONFIDENTIAL

Information classified Confidential - Do not copy (See last page for obligations)



### 13.2.1 Sequence of events: Mobile Originated–Short Burst Data message (MO-SBD)

1. User's application loads the mobile-originated-SBD message into the Q4000/QPRO.
2. Application instructs the Q4000/QPRO to send the SBD message to the Iridium Gateway.
3. Iridium Gateway SBD equipment:
  - receives the SBD message
  - sends an acknowledgement to the modem
  - creates an email message with the SBD data message as an attachment to the email.
4. Email message is sent to the destination email server hosted by the Value Added Reseller for processing of the data message.

### 13.2.2 Sequence of events: Mobile Terminated–Short Burst Data Message (MT-SBD)

1. Email message is sent to the Iridium Gateway server by the Value Added Reseller's host server.
2. Iridium Gateway SBD equipment receives the message and stores it in a database.
3. Modem initiates a "Mailbox Check" and the message is downloaded to the modem.
4. Modem sends an acknowledgement to the Iridium Gateway that the mobile-terminated-SBD message has been delivered.
5. Application extracts the mobile-terminated-SBD message from the modem and processes the message.

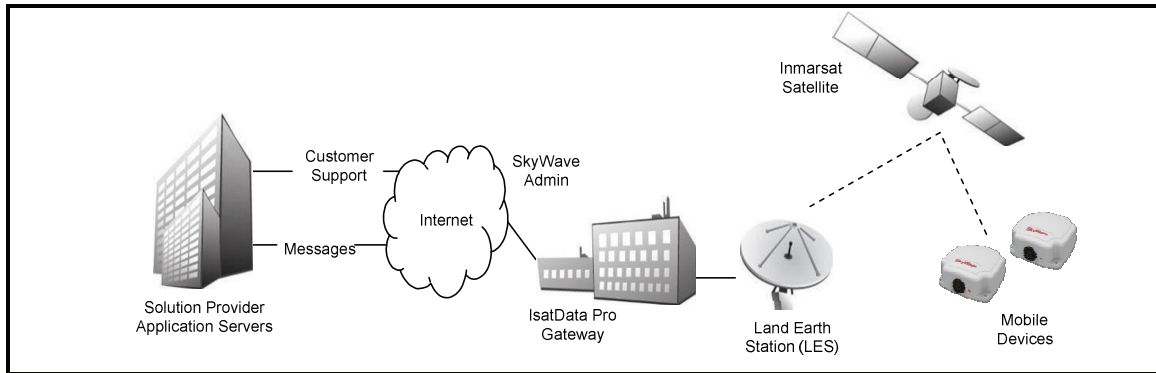
## 13.3 Inmarsat network

IsatData Pro is an Inmarsat service provided by SkyWave. IsatData Pro services use the Inmarsat satellite network, operating with four satellites. IsatData Pro provides bidirectional messaging services from a gateway to mobile devices via Inmarsat satellite services. IsatData Pro satellite service is a two-way, fully acknowledged communications protocol offering a relatively low data rate and near-real time data transfer.

IsatData Pro messaging capabilities make it an ideal service for applications requiring remote updates, form transfers, and text messaging combined with asset tracking. Typical applications include vessel and fleet management, and security, remote surveillance and telematics.

CONFIDENTIAL

Information classified Confidential - Do not copy (See last page for obligations)



**Figure 13-3: SkyWave's IsatData Pro network**

Key points of the system-level architecture:

- Message traffic sent to or from mobile devices passes through the SkyWave's IsatData Pro gateway to the Solution Provider's application servers.
- Mobile-originated messages may be from 2 to 6,400 bytes. Mobile-terminated messages may be from 2 to 10,000 bytes.
- Latency:
  - mobile-originated messages - 16 seconds for 100 bytes and 40 seconds for 1000 bytes
  - mobile-terminated messages - 15 seconds for 100 bytes and 45 seconds for 1000 bytes.
- Service activation, invoicing and customer support are provided through SkyWave.

CONFIDENTIAL

Information classified Confidential - Do not copy (See last page for obligations)

## 14 Event driven architecture

Event driven architecture is a significant element in the Q4000/QPRO user applications. Events have been defined which correspond to significant internal and external signals and conditions that are likely to appear. These events are then forwarded to the application.

Most actions the application takes in response to these events are handled by function calls. Many of the calls represent high-level actions, such as “send this message,” which abstract the low-level details. The Application Programming Interface (API) is set up so that these events are mapped to the application developer's ‘C’ code. The following sections describe the supported events and the circumstances under which they are posted.

The supported events of the Q4000/QPRO are:

- CAN\_MSG
- CELL\_NET\_IN\_VIEW
- CONTINUE
- COUNTER
- DIGITAL
- DIGITAL\_ALARM
- GLSS\_AVAILABLE
- MTS\_DTR
- MSG\_ACK
- MSG\_ALERT,
- MSG\_MID\_CHANGED
- MSG\_NAK
- MSG\_RCVD
- MSG\_SEND\_NAK
- NET\_CLEAR
- NMEA\_SENTENCE
- NO\_EVENT
- ORB\_ANTENNA\_VSWR
- POSITION\_FIX
- POSITION\_ALARM
- POWER\_ON
- RX\_MTS\_PKT
- RX\_SER\_PKT
- SAT\_IN\_VIEW
- SHUTDOWN
- SPEED\_ALARM
- TIMER
- TIME\_SYNC
- USER\_CMD

CONFIDENTIAL

Information classified Confidential - Do not copy (See last page for obligations)

A request for a CAN message with a specified Parameter Group Number (PGN) has been processed. A parameter [0-65535] value indicates the PGN of the message received. A second parameter value of 0 indicates the requested message was received; non-zero indicates an error, typically a timeout.

## 14.1 CELL\_NET\_IN\_VIEW

The cell network availability status has changed. Parameter value indicates:

- 0 – network is not available
- 1 – modem is registered on the GSM/GPRS network
- 2 - modem is connected to the GSM/GPRS network

## 14.2 CONTINUE

Used in Application Event Tables to indicate an additional Response Action for an event entry.

## 14.3 COUNTER

A software counter has reached 0, underflowed, or overflowed. The parameter indicates the counter number.

## 14.4 DIGITAL

A Digital measurement has completed, and the measured value is within limits specified by the Alarm High and Low values. The parameter [0-15] indicates which Sensor Table was used in making the measurement.

## 14.5 DIGITAL\_ALARM

A Digital measurement has completed, and the measured value violated limits specified by an Alarm High or Low value. The parameter [0-15] indicates which Sensor Table was used in making the measurement.

## 14.6 GLSS\_AVAILABLE

The Globalstar Simplex module availability status has changed: 0 means it is not ready for use; 1 indicates the module is working and ready to accept messages.

## 14.7 MSG\_ACK

Receipt of a MSG\_ACK event indicates that a message sent from the Q4000/QPRO was successfully delivered. The parameter [0-255] indicates the message reference number (or other tracking number if not an ORBCOMM message). This is currently supported for ORBCOMM and GSM/GPRS networks. Message reception is managed by @ref MSG\_RCVD.

## 14.8 MSG\_ALERT

A message is available from the network.

## 14.9 MTS\_DTR

The DTR Line on the MTS Serial Port has changed state. If the parameter [0-1] is zero, it indicates that the line has changed from High to Low Voltage; if one, it indicates the line has changed from a Low to a High voltage.

## 14.10 MSG\_MID\_CHANGED

The message is queued to a secondary network.

## 14.11 MSG\_NAK

The message was not sent by the primary network.

## 14.12 MSG\_RCVD

A terrestrial or satellite message packet was received from one of three sources, determined by parameter 1:

- TERR\_SMS: an SMS message was received
- TERR\_POP: a POP email message was received
- SATELLITE: a satellite message was received

The message length and data are contained in the incoming message.

## 14.13 MSG\_SEND\_NAK

The message was not sent and there will be no other attempts.

## 14.14 NET\_CLEAR

The network is cleared.

## 14.15 NMEA\_SENTENCE

The NMEA 0183 standard uses a simple ASCII, serial communications protocol that defines how data are transmitted. The data are passed in to the application as a message parameter, along with the length of the string as the parameter `msgLen`. The string is null terminated and can be displayed in a `printf()` statement.

**Note:**

The message packet should not be freed by the application; this is done by the foundation code.

CONFIDENTIAL

Information classified Confidential - Do not copy (See last page for obligations)



NMEA has the following application layer protocol rules:

- Each message's starting character is a dollar sign.
- The next five characters identify the talker (two characters) and the type of message (three characters).
- All data fields that follow are comma-delimited.
- Where data are unavailable, the corresponding field contains *NULL* bytes (e.g., in "123,,456", the second field's data are unavailable).
- The first character that immediately follows the last data field character is an asterisk.
- The asterisk is immediately followed by a two-digit checksum representing a hex number. The checksum is the exclusive OR of all characters between the \$ and the \*. According to the official specification, the checksum is optional for most data sentences, but is compulsory for RMA, RMB, and RMC (among others).
- <CR><LF> ends the message.

As an example, a waypoint arrival alarm has the form:

`$GPAAM,A,A,0.10,N,WPTNME*32`

where:

**Table 12-2: NMEA data format**

GP	Talker ID ( <i>GP</i> for a GPS unit)
AAM	Arrival alarm
A	Arrival circle entered
A	Perpendicular passed
0.10	Circle radius
N	Nautical miles
WPTNME	Waypoint name
*32	Checksum data

## 14.16 NO\_EVENT

Occurs periodically (once per second by default). An application can use this event to perform periodic status updates.

## 14.17 ORB\_ANTENNA\_VSWR

An ORBCOMM Antenna VSWR measurement result is available in 0.1 units (e.g. 15 => 1.5:1 VSWR).

## 14.18 POSITION\_ALARM

A GPS measurement completed and a Geo-Fence violation occurred. The parameter indicates which Sensor Table was used. This function is currently unsupported. Please contact QUAKE Global if you would like to implement GEO\_fencing in your product.

## 14.19 POSITION\_FIX

The GPS engine has just completed a position fix and the information is available. The parameter indicates which Sensor Table was used. In the current implementation, the parameter passed to `GPS_read` should always be zero, and the parameter returned on a `POSITION_FIX` event should always be zero.

## 14.20 POWER\_ON

This event occurs after reboot regardless of the cause of the power on/reset. It is the first event generated unless the power on was due to a wakeup timer. In that case, the `TIMER` event is the first event generated, followed by the `POWER_ON` event.

Application code executed after the `POWER_ON` event should perform initialization such as starting the appropriate network functions. A GPS read with a parameter of 0 is usually requested at the end of the initialization sequence.



**Note:**

Due to hardware requirements, if GSM/GPRS is powered down it must remain down for **at least** 15 seconds before being powered up again.

For more information, consult the `SYS_powerDown` description in the Application Programming Interface (API).

## 14.21 RX\_MTS\_PKT

An ORBCOMM OSI protocol packet was received from a serial port. The parameter indicates the type of packet received. Do not free the packet, as this is done by the caller.

## 14.22 RX\_SER\_PKT

A serial packet was received. The data are passed to the application as a message parameter, along with the length of the string. The string is null terminated and can be displayed in a `printf()` statement. The second byte of the message data indicates the type of packet received, either modem terminated (0x0C), User Command (0x0D) or ORBCOMM Globalgram (0x0E). For Iridium, raw data have been received.



**Note:**

The message packet should not be freed by the application; this is done by the foundation code.

## 14.23 SAT\_IN\_VIEW

The satellite-in-view status has changed. If the parameter [0-1] is zero, it indicates a satellite has gone out of view; if one, it indicates a satellite has come into view.

## 14.24 SHUTDOWN

System is shutting down; save the configuration parameters and exit.

CONFIDENTIAL

Information classified Confidential - Do not copy (See last page for obligations)