Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de la Telecomunicación

Software de monitorización de Smart Meter a través del módem MTX65i

Autor: Manuela Moreno Sánchez
Tutor: Germán Madinabeitia Luque

Dep. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2015

Departamento de Ingeniería Telemática

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Software de monitorización de Smart Meter a través del módem MTX65i

Autor:
Manuela Moreno Sánchez

Tutor:
Germán Madinabeitia Luque
Doctor Ingeniero de Telecomunicación
Profesor Colaborador

Dep. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2015

Trabajo Fin de Grado: Software de monitorización de Smart Meter a través del módem MTX65i

Autor: Manuela Moreno Sánchez

Tutor: Germán Madinabeitia Luque

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2015

El Secretario del Tribunal

A mi familia,
amigos y compañeros

Agradecimientos

Diciendo gracias a todos consigo que todas las personas que habéis sido no solo un apoyo, sino el aire que he respirado en los momentos de desaliento, seáis los primeros en ser mencionados en esta breve reseña de agradecimiento.

A mis padres, por enseñarme la constancia en el trabajo desde la infancia, haciendo que cada día diese gracias por todos los recursos que el mundo nos da, pero siendo siempre consciente de que sin trabajar la tierra nunca se consiguen los frutos.

A mis hermanas, por hacerme la persona que soy hoy, por educarme cuando no había nadie más para hacerlo, por apoyarme cuando nadie más confiaba en mí, y por alentar sueños que quizá nunca alcance.

A mi compañero, por transmitirme que las cosas tienen solución o no, pero que si la tienen, tarde o temprano la encontraremos, con más o menos suerte, con más o menos paciencia. Y por estar.

A mis amigos y amigas, que han sabido soportarme y hacerme reír en cualquier momento, y que no han salido corriendo ni se han olvidado de mí cuando parecía que yo sí me había olvidado de ellos.

A mis profesores, que desde pequeña no solo me han enseñado, sino que me han valorado, siendo un ejemplo y objeto de mi admiración.

A todo el departamento y en especial a mi tutor, por facilitarme conocimientos necesarios para intentar comprender el mundo que me rodea, por exigirme con paciencia y darme una oportunidad de mejora en cada fracaso.

A todas las personas, positivas o negativas, que han estado en mi vida durante estos años. Las primeras por aportar luz en cualquier momento. Las segundas por ser mi motivación para llegar más allá.

Ojalá todas las personas aquí mencionadas reciban lo que yo he recibido de ellas.

Resumen

En la actualidad, la cantidad de información generada a diario suscita un reto tanto a nivel de procesamiento como de recolección de la misma. Por ello es necesario que las redes de telecomunicaciones converjan a un sistema de comunicaciones complejo que abarque las emergentes fuentes de información y transmita la información íntegra y segura hasta los centros de procesamiento y control de datos.

Durante los últimos años se está evolucionando de manera natural al modelo de redes IoT, en el que elementos que anteriormente no producían ninguna información ahora sí lo hacen, de manera que ésta se encuentre disponible en el menor tiempo posible.

A este respecto, una de las líneas de desarrollo del concepto es la de mejorar las comunicaciones de datos de dispositivos que ya cumplen con el propósito de recolectarlos, como es el caso de un contador de medidas de consumo eléctrico. La cantidad de estos dispositivos y su localización convierten en una tarea tediosa la recogida de la información y por este motivo en los últimos tiempos se están sustituyendo dichos elementos por contadores de telegestión que permitan la interacción con el aparato de medida a distancia desde una central.

El presente proyecto tiene como objetivo cumplir con la adaptación y mejora utilizando un equipo intermedio que actúe como pasarela entre los registradores y las centrales de información, para así evitar en algunos casos el cambio de contador que comprende un mayor coste económico que el del terminal módem utilizado como intermediario.

Abstract

Currently, the amount of information generated daily poses a challenge both at the processing level as its compilation. Therefore it is necessary that telecommunications networks converge on a complex communications system which includes emergent information sources and conveys complete and reliable information to processing centers and data control. Thus, during the last several years it is evolving naturally to IoT network model in which many elements that previously did not produce any information now they do, and whose purpose is to be available in the shortest possible time.

On this point, one of the development lines of this concept is to improve data communications of devices which already comply with the purpose of collect them, in our case a counter of measures of electric consumption. The amount of these devices and their location make of collecting information a tedious work and therefore in recent years are being replaced by remote control counters which allow interaction with the remote measurement instrument from a central.

This project aims to comply with the adaptation and improvement by using an intermediate equipment which serves as a gateway between registrars and information centrals in order to avoid in some cases the counter change that comprises a higher cost than the one of modem terminal used as an intermediary.

Índice

ÍNDICE DE ILUSTRACIONES

1 INTRODUCCIÓN

Este proyecto ha sido propuesto por la empresa Wellness Telecom desde el área Wellness Smart, la cual se encarga del desarrollo de los proyectos relacionados con las denominadas "ciudades inteligentes" que mediante el uso de las nuevas tecnologías pretenden mejorar nuestro entorno y día a día.

La idea del proyecto es realizar un software que permita que el terminal módem MTX65i se comunique bidireccionalmente tanto con un contador de medidas eléctricas como con el centro de recolección de datos. Para ello se utilizarán comunicaciones a través del puerto serie en el binomio módem-contador, y GPRS en el caso del módem y la central.

La principal ventaja de la incorporación del módem es que, a bajo coste, podemos conseguir una pasarela de comunicación entre un dispositivo que de otro modo no sería capaz de transmitir los datos obtenidos sin ser sustituido por otro compatible.

Para cumplir este cometido se necesita una aplicación que se ejecute en el módem gracias al firmware que contiene basado en Java ME (versión reducida de la máquina virtual de Java para sistemas embebidos). Esta aplicación se encargará de implementar las comunicaciones necesarias entre un contador compatible, el módem y el servidor.

El contador utilizará el protocolo DLMS [1] o Modbus [2] en un principio, aunque podría ser implementado para todos los contadores que cumplan el estándar IEC 60870-5, adaptando únicamente la trama esperada por el módem y la comunicación del contador.

Para la codificación y pruebas será necesario un ordenador con el sistema Windows XP o 7 con el IDE de Eclipse y el plugin Mobile Tools for Java para el desarrollo de aplicaciones MIDlet. También contará con el SDK proporcionado por el fabricante correspondiente al TC65i que es el módulo interno del módem. Dentro del mismo se incluye el driver que posibilita la detección del dispositivo al ser conectado mediante USB.

De manera puntual, para las comunicaciones serie se utilizará el sistema operativo Linux Ubuntu y Minicom a través de una terminal para establecer la comunicación serie RS232.

La central deberá implementar un servidor que atienda las peticiones HTTP o de conexión del módem para la recepción de medidas el cual implementaremos mediante una instancia virtual de Ubuntu con Apache y PHP así como la posibilidad de comportarse como cliente en el caso de solicitar una medida puntual en un momento indeterminado.

Gracias a todos estos elementos diseñaremos y desplegaremos un software autónomo capaz de enviar medidas a través de la red de forma segura y reduciendo los costes que supone la sustitución de un nuevo equipo de telegestión.

1.1 Objetivos

El principal objetivo de este proyecto es conseguir una aplicación software autónoma instalada en el módem que de manera periódica solicite las medidas tomadas por el contador y acto seguido, las envíe a través de GPRS utilizando conexiones TCP vía socket o con HTTP Get o Post.

Así mismo, otro objetivo crucial es que dada la futura localización del módem, se puedan establecer conexiones externas de manera segura con el terminal para solicitar una lectura asíncrona de las medidas del contador que se envíe de manera inmediata al solicitante.

Esto supone el principal reto dada la naturaleza del dispositivo, ya que el módem cuenta con un firmware basado en Java ME diseñado para aplicaciones sencillas en sistemas embebidos.

El tercer objetivo es la comunicación con el contador de manera que se establezca un protocolo de comunicación en el que a solicitud del módem, el contador devuelva los datos en un formato determinado de trama que permita su mínimo procesamiento antes de ser enviada a la central.

Dado que para lograr la comunicación con el exterior vía GPRS es necesario contratar un proveedor de servicio que nos suministre el acceso a la red de datos, lo ideal es que la comunicación de éstos consuma lo menos posible y para ello es necesario que el tamaño de los datos a envía no supere ciertos límites con lo que además disminuiremos el número de errores en la transmisión o recepción.

Tampoco podemos olvidar que el módem tiene una capacidad computacional reducida (400KB de RAM) lo que nos obliga a utilizar Java Micro Edition que es una versión reducida cuya librería cuenta con un número de clases muy escaso, entre las que se incluyen las que permiten el uso de comandos destinados al módem por lo que otro de los objetivos sería que la aplicación fuese lo más eficiente posible teniendo en cuenta la herramienta utilizada. Pero a su vez debemos conseguir que dos procesos independientes como son la realización de solicitud de medidas síncronas al contador y la permanente escucha a la espera de solicitudes por parte del servidor central se ejecuten con normalidad y que interaccionen entre sí llegado el momento.

También sería conveniente lograr la menor probabilidad de error y si éste se produjese, que pueda ser solventado mediante una nueva versión desplegada vía OTAP. Y del mismo modo si ocurre algún fallo en la alimentación del dispositivo cuando ésta se restablezca que tanto el módem como la aplicación reinicien su funcionamiento con total autonomía sin que un agente externo intervenga en el proceso.

Por último se fijará como objetivo dada la naturaleza del dispositivo y del lenguaje de programación utilizado que la plataforma sea lo más escalable posible y compatible con el mayor número de dispositivos. En caso de necesitar adaptación se promoverá que tenga el menor impacto posible.

1.2 Alcance

El alcance de este proyecto comprenderá prácticamente la totalidad de los objetivos marcados en el punto anterior, salvo los relacionados con la comunicación con el contador dado que este dispositivo no ha sido proporcionado aún por la empresa. Esto implica que la parte de la comunicación serie entre el contador y el módem se realizará de forma simulada conectando el módem a un ordenador con puerto serie en su lugar y enviando y recibiendo datos por esta conexión. De igual modo ocurre con la autenticación con el contador cuya implementación no ha sido posible.

En definitiva se alcanzará el objetivo final de comunicación entre los tres elementos (módem, servidor y contador) pero emulando la parte del contador mediante un ordenador conectado al otro lado de la comunicación serie.

1.3 Estructura de la memoria

Este apartado tiene como objetivo dar una vista preliminar del documento al completo de manera que el lector pueda navegar con menor dificultad en el caso de buscar extractos concretos.

1. Introducción

Es el apartado en el que nos encontramos, que proporciona información general acerca del desarrollo del proyecto así como una descripción de los objetivos identificados y el alcance del mismo.

2. Estado del arte

Pretende informar sobre algunas de las soluciones similares que se han implementado o se están desarrollando actualmente relativas a las comunicaciones entre dispositivos de telemetría y los sistemas centrales que necesitan sus medidas, como puede ser el caso del software MTX Tunnel o el proyecto Meters and More.

3. Descripción de la solución

Se dará una primera visión de la solución que se desea implementar para cumplir con los objetivos y el alcance descritos anteriormente.

4. Herramientas

En este extenso apartado detallaremos todas las herramientas utilizadas a lo largo del proyecto.

Comenzaremos por los dispositivos físicos que se utilizarán como son el módem MTX65i, una tarjeta SIM con acceso a servicio de datos, el contador de medidas eléctricas, un ordenador con SO Windows 7 y otro con Linux Ubuntu.

También se mencionarán los lenguajes de programación para la implementación como son Java en su versión JME y los comandos Hayes (conocidos también como comandos AT).

De igual modo se explicarán los protocolos fundamentales como TCP en el caso de sockets para el envío de la información capturada por el módem o HTTP si elegimos trabajar a nivel de aplicación con el mismo propósito de envío. Se hará una mención a DLMS [1] o Modbus [2] como protocolos para la lectura de datos del contador y el uso de RS232 para la comunicación del puerto serie.

En cuanto al software necesitaremos que el módem cuente con la versión de firmware adecuada, un cliente como putty para establecer y emular conexiones serie si nos encontramos en el SO Windows y Minicom en el caso de Ubuntu. Para el desarrollo y la codificación utilizaremos el IDE de eclipse, con plugins adicionales, además del SDK del módulo TC65i que contiene el módem.

Por último para las pruebas será necesario levantar un servidor Apache con PHP sobre Ubuntu 14.04 gracias a una instancia virtual en la nube y tener acceso a la VPN para realizar las comunicaciones en la intranet de la empresa.

5. Preparación del entorno

Aquí se proporcionará información detallada de todas las instalaciones de software, puesta en marcha de dispositivos y pruebas de comunicaciones realizadas.

Finalmente se incluirá un breve apartado de los problemas encontrados en la instalación y las soluciones aportadas.

6. Programación de aplicaciones MIDlet

Este apartado consistirá en una inmersión en la programación de aplicaciones MIDlet que utilizan java como base con una versión reducida (JME) pensada especialmente para dispositivos con pocos recursos. También se incluirán algunos ejemplos de aplicaciones sencillas como son un Hello World o un Timer.

7. Desarrollo de la solución

La arquitectura y disposición del hardware, así como el desarrollo software de la aplicación serán detalladas en este apartado. Se explicará como a través de una programación modular se logran por separado los hitos marcados. Por ejemplo para un hito como la comunicación de la información usando HTTP vía Post, se crea un módulo que únicamente realiza esta función y cuando ésta funciona, se añade al resto de módulos ya implementados como puede ser Serial Port de manera que vamos construyendo la aplicación final.

8. Pruebas

Se describirán las pruebas realizadas a la aplicación por la interacción entre el módem y las herramientas destinadas a la comprobación del funcionamiento y el cumplimiento de los requisitos especificados.

9. Resultados

Exposición de los resultados obtenidos al realizar las pruebas así como en la puesta en funcionamiento de la aplicación desarrollada.

10. Conclusiones y propuestas de mejora

Se detallarán las conclusiones extraídas de todo el proyecto en su conjunto, incluyendo un subapartado con las vías de mejora propuestas sobre la aplicación funcional inicialmente desplegada.

11. Bibliografía, Referencias y ANEXOS

El contenido de estos apartados contará con una relación de las fuentes de información utilizadas, así como de un conjunto de anexos como puede ser una lista de los comandos AT utilizados durante el desarrollo.

2 ESTADO DEL ARTE

Dada la vital importancia de la recolección de datos sobre los suministros de servicios (en cuanto a facturación y demanda), el desarrollo de soluciones similares que cumplan los objetivos aquí marcados es algo que ya se está implementando con soluciones similares a la que pretendemos alcanzar en este proyecto.

2.1 MTX Tunnel [3] [4]

Directamente relacionada con el dispositivo que vamos a utilizar de sistema intermedio (módem MTX65i), existe una aplicación denominada MTX Tunnel.

Se trata de un software que se incluye en los modelos de módem MTX [5], cuya funcionalidad radica en servir de pasarela transparente GPRS/GSM-Serie (RS232/RS485) [6] de manera que los equipos que dispongan de puerto serie pueden ser gestionados remotamente. Basta con realizar una conexión TCP/IP con el MTX-Tunnel y todo lo enviado por dicha conexión acabará recibiéndose en el puerto serie del dispositivo, y por ende, al equipo conectado al otro lado que no tiene porqué incluir implementar MTXTunnel.

Se trata de un software propietario cuya licencia ha de ser adquirida previo pago, siendo un incentivo en la búsqueda de una aplicación alternativa a la desarrollada por MTXTunnel para el MTX65i en este proyecto.

2.2 Meters and More [7]

Es una organización internacional que promueve la utilización del protocolo del mismo nombre para su inclusión en los nuevos contadores de tele gestión que se están instalando en países como España e Italia a través de Endesa [8] y Enel. El protocolo permite la transferencia bidireccional de datos desde el contador inteligente hasta la central.

En su especificación [9] indican que su protocolo se adapta a las características que el estándar de CENELEC [10] [11] propone con el objetivo de desplegar este protocolo como el principal a la hora de las comunicaciones para extraer datos de contadores de telegestión.

2.3 4CTT [12]

Es un concentrador de comunicaciones que forma parte de un Sistema de telegestión con lectura automática de contadores (AMR) a través de la propia red de baja tensión.

El dispositivo forma parte de un sistema compuesto por un grupo contadores residenciales o industriales con comunicación por la red de baja tensión, un sistema de supervisión de baja tensión y el propio sistema de tele gestión. Utiliza el protocolo RS232 a la par que Ethernet para las comunicaciones.

En esta línea podemos encontrar muchos ejemplos de empresas que actualmente están desarrollando soluciones de tipo Smart Grids. [13] [14].

3 DESCRIPCIÓN DE LA SOLUCIÓN

La solución a implementar consiste crear una aplicación MIDlet que utilice la conexión física del módem al contador para solicitarle las medidas de manera síncrona y que posteriormente sean enviadas al servidor que espera la información. De forma simultánea a la tarea periódica de lectura es necesario que el módem permanezca a la escucha de una posible conexión entrante desde el servidor central para una recolección asíncrona de datos de lectura del contador.

En el siguiente esquema tenemos una primera aproximación de la secuencia de acciones que la aplicación debe implementar:

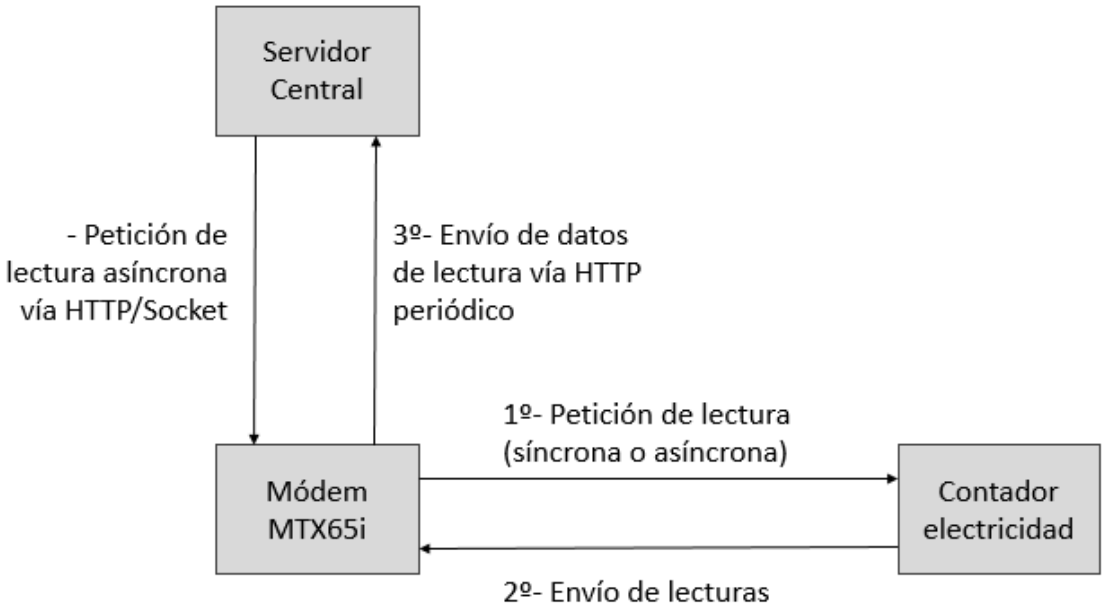

Ilustración 3-1. Esquema secuencial de las acciones de la aplicación

La conexión entre el contador y el módem será en principio de interfaz de puerto serie a interfaz de puerto serie gracias a un conector DB9 en cada extremo por lo que ambos deben encontrarse próximos el uno del otro. Sin embargo, en el caso del servidor y dado que es el principal objetivo de esta aplicación, la comunicación utilizará GPRS para transportar la información hasta el servidor y recibir la consecuente respuesta a través de internet como se puede ver en el siguiente esquema:

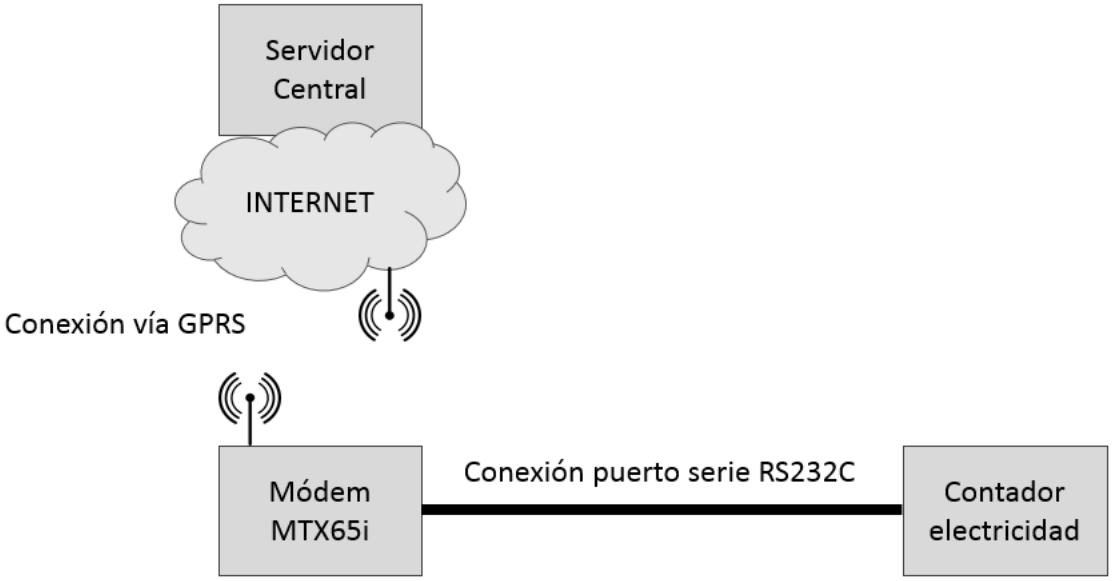

Ilustración 3-2. Esquema de las conexiones entre los dispositivos

Por último, los protocolos utilizados en las comunicaciones entre los dispositivos serán en el caso del módem y el contador el que determine el contador, como por ejemplo DLMS [1] o Modbus [2].

En el módem si es éste el que envía los datos utilizará HTTP vía Post. Sin embargo, si se trata del servidor el que de manera asíncrona establece una conexión para solicitar una nueva medida se utilizará un socket TCP debido a que solo es posible la creación de un socket de este tipo en modo servidor. Si la medida no pudiera ser enviada al servidor o si éste no la recibiera correctamente la guardaremos en un fichero que se almacenará en la memoria del dispositivo para su posterior recuperación. Todo esto queda reflejado en el siguiente esquema:

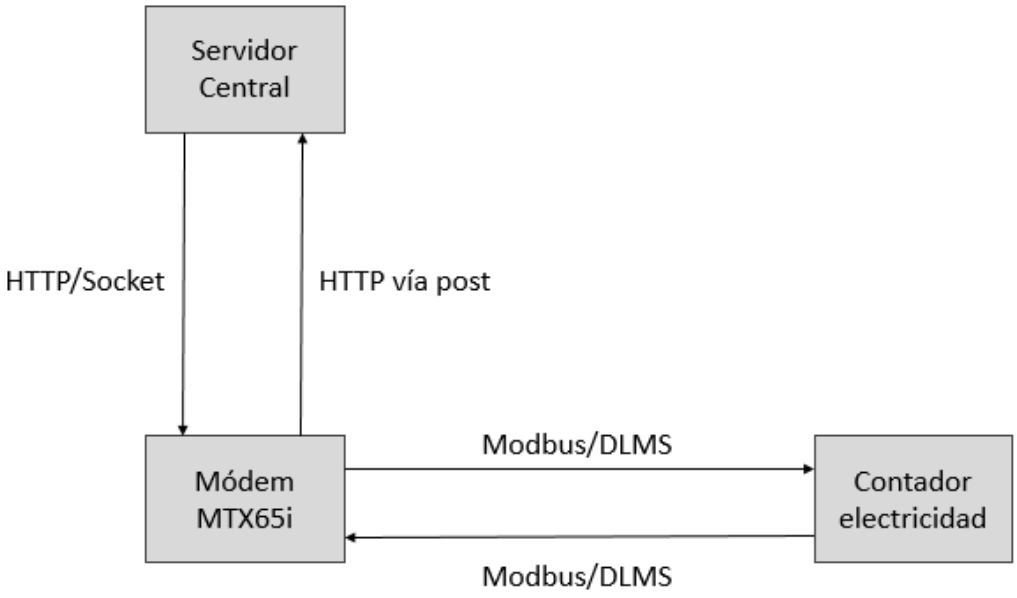

Ilustración 3-3. Esquema de los protocolos para el envío de la información

4 HERRAMIENTAS

Este apartado pretende explicar en detalle los dispositivos físicos utilizados, el software instalado cuyo procedimiento se detallará en el apartado Preparación del entorno gracias al cual conseguiremos la aplicación MIDlet final del módem.

4.1 Dispositivos utilizados

4.1.1 Módem MTX65i [15]

El modem MTX65i contiene como módulo integrado el TC65i [16] de donde toma su nombre. Es éste el que le proporciona la posibilidad de utilizar conectividad con TCP/IP a través de AT y JVM (Máquina Virtual de Java), y las características del M2M (Machine to Machine) [17].

Permite la posibilidad de usar las bandas de radio 850/9000/1800/1900 MHz, además de un conector RF y enlace estabilidad radio (RLS).

Tiene 400 KB de RAM y una memoria de 1.7 MB, suficiente para albergar el ejecutable de la aplicación.

Cuenta con un conector de 80 pines B2B conector Micro USB, I2C Y SPI. Una de las mejoras respecto al TC65 es que incorpora dos interfaces serie RS232485 denominadas ASC0 y ASC1. También tiene acceso SIM remoto y antena externa.

El proveedor suministra el cable conversor Micro USB/USB, el cable de alimentación y una antena extraíble.

Ilustración 4-1. Módem Siemens MTX65i

Ilustración 4-2, Antena externa

Ilustración 4-4. Cable de alimentación

Ilustración 4-3. Conversor MicroUSB/USB

Las interfaces serie del módem quedan representadas en la siguiente imagen:

Ilustración 4-5. Interfaces serie del módem

4.1.2 Contador Cirwatt B200RCP [18]

Se trata de un contador monofásico digital multifunción debido a que es capaz de tomar medidas tanto de energía activa como reactiva. Cumple la normativa europea actual vigente en contadores de energía (MID) EN 50470-1 y EN 50470-3 lo que permite su instalación en cualquier país de la comunidad europea.

Ilustración 4-6. Contador Cirwatt B200RCP

Dispone de comunicaciones PLC / PRIME (Power Line Carrier) [19] a través de la red eléctrica así como de

puerto óptico.

Estas comunicaciones se realizan gracias al protocolo DLMS [1]. Dispone igualmente de un registrador con capacidad para información de hasta 3 meses de registros horarios, de los 6 tipos de energía. Del mismo modo dispone de la habilidad de hacer lecturas de datos en ausencia de tensión. Incorpora un elemento de corte que permite al usuario controlar la demanda del suministro para ser gestionada de manera remota utilizando comunicaciones PLC.

El cometido principal de este contador CIRWATT B es la medida de energía activa y reactiva para facturación. La comunicación PLC permite la descarga a distancia de todos los datos obtenidos por el contador a través del concentrador PLC1000 u otro concentrador PRIME. El elemento de corte integrado en el contador permite la gestión a distancia del suministro además de la posibilidad de ser programada para que al alcanzar la potencia contratada se abra el circuito para su posterior reconexión, garantizándose la seguridad del usuario.

Para información adicional se puede consultar la ficha técnica. [20]

Debido a problemas logísticos aún no está disponible el contador para completar el montaje.

4.1.3 Ordenador HP ProBook 4520s [21]

Las características hardware principales del portátil es que dispone de tres puertos USB pero no dispone del conector DB9 necesario para las comunicaciones del puerto serie. Por este motivo será necesario utilizar un ordenador de sobremesa cuya CPU disponga del mismo. En nuestro caso utilizaremos uno de los ordenadores disponibles en el departamento de telemática.

El portátil, por imperativo de las herramientas de desarrollo que se utilizan para el dispositivo tiene instalado el SO Windows 7. El ordenador de sobremesa del departamento dispone de Ubuntu 14.04.

4.1.4 Tarjeta SIM con tarifa de datos

Dado que uno de los objetivos principales es la comunicación de los datos recogidos vía GPRS es imprescindible contar con una tarjeta insertada en el modem de manera que tengamos acceso a la red de datos (en esta solución no es necesario realizar llamadas). Se puede utilizar una tarjeta de cualquier operador nacional, aunque se recomienda utilizar en principio uno que no sea Operador Móvil Virtual [22].

Tampoco es requisito indispensable que la tarjeta tenga código PIN. Sólo se necesita configurar de la manera correcta el módem para que tenga acceso a la red mediante el APN correspondiente que en nuestro caso se trata de un APN perteneciente a una empresa privada.

La configuración para la conexión vía GPRS se detallará en el apartado Primeros pasos con el módem MTX65i.

4.2 Lenguajes de programación

4.2.1 Comandos Hayes (Comandos AT) [23] [24] [25]

El conjunto de comandos Hayes [26] (también denominado comandos AT) es un lenguaje desarrollado por la compañía Hayes Communications [27] que posteriormente se convirtió en estándar abierto de comandos para configurar los parámetros de módems. El nombre de comandos AT se debe a que todos los comandos están precedidos por estas dos letras que significan "Attention".

El módem puede encontrarse en dos modos:

- Modo de datos: el módem trata todo lo que recibe como datos y los envía a través de la línea.

- Modo de comandos: los datos recibidos se interpretan como comandos que deben ser ejecutados por el módem en cuestión.

Para cambiar del modo de datos al de comandos hay una secuencia de escape ("+++") seguido de una pausa de un segundo [28].

Para regresar al modo de datos se envía "*O*" aunque en realidad es raro utilizar el comando de línea ya que tras los comandos puede configurarse para que vuelva al modo de datos.

Las principales funciones de los comandos AT son manipulaciones de línea, marcado y colgado, así como controles de configuración como el baud rate o tasa de símbolos por segundo.

El conjunto de comandos AT se puede dividir en:

- Comandos básicos: Se componen por AT seguidos de una letra y opcionalmente (en el caso de que haya varios que hagan referencia al mismo comando) un número. Por ejemplo, con ATE1 habilitamos el eco de los comandos que enviemos para que se muestren por la pantalla de la interfaz de conexión que estemos utilizando, mientras que con ATE0 lo deshabilitaríamos.
- Comandos especiales: Se escriben AT& seguido de una letra. Por ejemplo AT&F restaura los valores de configuración de fábrica. Ocurre igual con AT\, que repite el último comando ejecutado.
- Comandos extendidos: Vienen definidos por el fabricante del módem. En nuestro caso pueden ser del tipo AT+ seguido de unas siglas que ayudan a conocer la función del comando, como por ejemplo, AT+COPS que se utiliza para seleccionar el operador. Para comandos sobre parámetros de configuración lo habitual es que aparezca un ^ en lugar del signo +. Un ejemplo de este comando sería AT^SCFG que permite ver o modificar los parámetros de la configuración extendida.

Además, dependiendo de la naturaleza de cada comando, éstos se podrán ejecutar en 4 modos distintos:

- Modo consulta. En este modo podremos ver los posibles valores que podemos asignar al comando. Se ejecuta insertando el comando seguido de "=?". Un ejemplo sería AT+IPR=?, que nos devolvería la lista de tasas de bit que podemos utilizar en el modem.

```
AT+IPR=?
+IPR: (1200,2400,4800,9600,14400,19200,28800,38400,57600,115200,230400,460800),(
300,600,1200,2400,4800,9600,14400,19200,28800,38400,57600,115200,230400,460800,9
21600)
```

Ilustración 4-7. Captura de la ejecución del comando AT+IPR=?

- Modo lectura. En este caso se ejecuta el nombre del comando seguido de un signo "?". De esta manera obtendremos el valor actual del comando. Por ejemplo el comando anterior sería:

```
AT+IPR?
+IPR: 115200

OK
```

Ilustración 4-8. Captura de la ejecución del comando AT+IPR?

- Modo escritura. Gracias a este modo podemos asignar un valor de la lista de valores soportados por cada

comando (que podemos conocer ejecutando el modo consulta). Por ejemplo, para asignar un valor a la tasa de bits que utilizará el módem:

```
AT+IPR=9600
OK
```

Ilustración 4-9. Captura de la ejecución del comando AT+IPR

- Modo ejecución. Ciertos comandos solo pueden ser ejecutados puesto que no tiene utilidad alterar su valor, basta con que realicen la acción para la que están definidos. Es el caso del comando ATI, que nos muestra información acerca del dispositivo.

```
ATI
Cinterion
TC65i
REVISION 02.004

OK
```

Ilustración 4-10. Captura de pantalla de la ejecución del comando ATI

Por último cabe destacar que, al ejecutar un comando de escritura, los cambios que apliquemos pueden aplicarse inmediatamente y quedar registrados en la memoria no volátil del dispositivo, aplicarse tras el reinicio del terminal y quedar registrados en la memoria no volátil del dispositivo, o aplicarse con carácter inmediato pero que dejen de tener efecto si se deja de alimentar el terminal.

En el caso de que queramos guardar los valores para la configuración que hayamos dado en un momento determinado como un perfil propio de configuración para restaurarlos al reiniciar el terminal debemos ejecutar el comando AT&W que almacena estos ajustes en la memoria no volátil. Para restaurar esta configuración al encender el dispositivo basta con ejecutar AT&Z y si queremos recuperar los valores de fábrica podemos utilizar AT&F.

En el anexo Relación de Comandos AT relevantes se pueden ver una relación de los comandos AT más utilizados y para una consulta más completa el pdf con el manual de comandos AT disponible en el SDK del módulo TC65i.

4.2.2 Java [29]

Es el lenguaje de programación orientada a objetos por excelencia.

Se trata de un lenguaje de propósito general basado en clases y diseñado para tener el menor número de dependencias de implementación (aunque deriva de C y C++ tiene un menor número de instalaciones de bajo nivel que éstos).

Su objetivo es permitir a sus desarrolladores programar una sola aplicación que se pueda ejecutar en cualquier dispositivo sin necesidad de volver a ser compilado (Write Once Run Anywhere) [30].

Es ampliamente utilizado para aplicaciones web cliente servidor [31].

Actualmente Oracle Corporation es la propietaria de la aplicación oficial de la plataforma Java SE. La implementación de Java carece de normativa al respecto por parte de la ITU, IEC, etc., por lo que la ofrecida por Oracle es el estándar de facto.

Existen dos distribuciones: Java Runtime Enviroment (JRE) que contiene lo necesario para ejecutar programas Java y está destinado a usuarios finales, y el set de desarrollo Java Development Kit (JDK) que contiene además herramientas de desarrollo como el compilador de Java [32], Javadoc [33], un depurador, etc.

4.2.2.1 JME [34]

Conocido anteriormente como Java 2 Micro Edition, JME es una derivación de Java que proporciona un entorno para las aplicaciones que se ejecutan en sistemas embebidos y dispositivos móviles (sensores, teléfonos móviles, etc.)

Incluye interfaces flexibles, una función de diversos protocolos y soporte para aplicaciones en red y fuera de línea. Estas aplicaciones son compatibles con multitud de dispositivos pero a la vez tratan de aprovechar los recursos de cada uno de ellos.

En nuestro caso Java incluye Java ME SDK para desarrollar aplicaciones con la configuración CLDC (Connected Limited Device Configuration) [35] que define el conjunto base de las interfaces de programación gracias a las cuales se construyen aplicaciones en dispositivos móviles como es nuestro caso. Combinado con un perfil Mobile Information Device Profile (MIDP) como el Information Mobile Profile – Next Generation (IMP-NG) [36] proporciona una sólida plataforma java para la ejecución de aplicaciones en sistemas embebidos.

La empresa Cinterion ha desarrollado una serie de librerías para esta plataforma que se incluyen en el SDK del TC65i y que complementan la programación de aplicaciones MIDlet.

4.3 Comunicaciones

4.3.1 Comunicación serie RS232 [37]

La comunicación serie RS232 es la base de la transmisión de datos entre el módem MTX65i y el contador de medidas eléctricas.

Se trata de un estándar eléctrico y mecánico de los sistemas de transmisión de información [38]. Aquí se especifica el método de conexión entre los dispositivos equipo terminal de datos (DTE) y equipo de terminación del circuito de datos (DCE) que en su origen fue desarrollado para interconectar terminales de datos a los módems [39].

Este estándar cuenta con tres partes principales: características eléctricas de la señal, las características mecánicas de la interfaz y la descripción funcional de los circuitos de enlace [40].

RS-232 define los niveles de tensión y las características que debe tener la puesta a tierra, en referencia a las características eléctricas de la señal que en nuestro caso vendrán definidas por las características dadas por los equipos que intervienen en la comunicación (módem y ordenador de sobremesa) que cumplen con el estándar. También se describen los circuitos asociados a un sistema desequilibrado.

En cuanto a las características mecánicas de la interfaz entre el DTE y el DCE se indica que la toma de corriente normalmente estará en el DCE y que se utilizará un cable con conectores DB9/DB25. En nuestra aplicación utilizaremos un conector DB9 entre el módem (DCE) y el equipo (DTE).

Ilustración 4-11. Cable con conector DB9 Macho

Sobre la descripción funcional podemos decir que el protocolo permite fijar los bits que serán identificados como datos, si queremos bit o no de paridad y si utilizaremos bit de parada.

4.4 Protocolos

4.4.1 TCP [41]

El Protocolo de Control de Transmisión busca dar fiabilidad en la capa de transporte del modelo OSI entre los equipos que establezcan conexiones en internet.

Se trata de un protocolo fiable, orientado a conexión extremo a extremo. Está diseñado para que encaje en el modelo de capas OSI, concretamente en la capa de transporte que es la capa que soporta a la de aplicación y justo por encima de la capa de red, que proporciona a TCP un camino para el envío y recepción de segmentos de longitud variable según el canal.

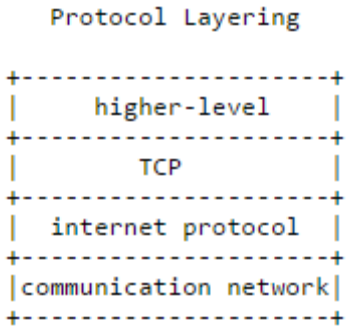

Ilustración 4-12. TCP en el modelo de capas OSI

En nuestro caso TCP puede utilizarse como alternativa a la solución de peticiones y respuestas HTTP diseñadas para que la aplicación comunique los datos recopilados o como soporte del socket servidor.

4.4.2 HTTP [42]

El Protocolo de Transferencia de Hipertexto se diseñó para la distribución de información entre sistemas conectados a través de internet, encontrándose en la capa de aplicación. El protocolo se utiliza también en servidores de nombres o sistemas de gestión de distribución de objetos gracias a sus métodos de petición, códigos de error y encabezados.

Una de las características de HTTP es la negociación de la representación de los datos, permitiendo que sean los sistemas a la hora de transferir la información los que fijen este aspecto sin venir prefijado anteriormente.

Se basa en la disciplina que le proporciona el Identificador Uniforme de Recursos (URI) [43], a través de una ubicación (URL) o un nombre (URN) para indicar al recurso al que se va a aplicar el método que se indique.

HTTP se utiliza como protocolo genérico para la comunicación entre los agentes de usuario y proxis/entrada a otros sistemas de Internet incluyendo SMTP [44], FTP [45], etc.

Algunos aspectos básicos a tener en cuenta sobre HTTP son los términos que lo definen:

- Conexión: Se trata de un circuito virtual que la capa de transporte establece con el propósito de llevar a cabo comunicación.
- Mensaje: Unidad básica de comunicación en HTTP que consiste en una secuencia estructurada de octetos que cumplen con la sintaxis del protocolo [35].
- Request: Mensaje que contiene un HTTP Request con los campos que se definen en la RFC [35].
- Response: Mensaje que se envía en respuesta al mensaje anterior con los campos definidos [35].
- Cliente: Programa que establece conexión con el propósito de enviar peticiones.
- Agente de usuario: El cliente que inicia la petición. Suelen ser navegadores, editores, etc.
- Servidor: Programa que acepta conexiones con el fin de dar respuestas a los clientes. Cualquier programa puede actuar como servidor al ser el que envíe las respuestas.

Según donde se encuentre la información (parte de la URI o no) se definen los principales métodos: GET y POST que aportan el primer paso de seguridad dentro del protocolo.

Además en esta línea podemos mencionar la mejora a HTTPS que establece una conexión cifrada segura para este protocolo.

En nuestro caso tendremos que el cliente será el módem que de manera periódica enviará un HTTP Request con la información obtenida del contador vía POST con el formato definido en el siguiente apartado.

4.4.2.1 Mensaje de datos enviados por HTTP

Los datos con la información sobre las medidas realizadas se transmitirán en un mensaje de tipo HTTP utilizando el método Post para evitar que la información viaje en la cabecera y lo haga en el cuerpo del mensaje.

El formato del mensaje, además de los campos obligatorios como Content-Length o el Content-Type, incluiría en el cuerpo la cadena *data=* seguida de dos parámetros especificados por las claves *measure* y *time* separados por comas, cuyo valor nos proporcionará el contador mediante la comunicación serie.

Por tanto el cuerpo del mensaje quedaría de la siguiente manera:

measure=value&time=valueTime

En nuestro caso utilizaremos como se puede comprobar en el anexo del código implementado el Content-type */x-www-form-urlencoded* que es el necesario para que la petición HTTP sea procesada correctamente por nuestro servidor y recibamos el correspondiente eco. El content-type no tiene por qué ser el mismo en todos los casos, también se puede utilizar texto plano por ejemplo.

Los valores se envían en el cuerpo de la solicitud, en el formato que especifica el tipo de contenido.

Los métodos utilizados en JME para añadir el contenido de los campos necesarios incluyendo el body se especificarán en el apartado referente al desarrollo del módulo HTTP vía POST Module.

4.4.3 Modbus [46]

Dado que por motivos ajenos al desarrollo de este proyecto no se dispone del contador con el se iba a establecer la comunicación de los datos para su transmisión, la parte referente al protocolo utilizado para la petición y respuesta de datos entre el módem y el contador se va a emular de la manera que se explica a continuación.

Con objeto de simplificar esta emulación y puesto que se trata de un protocolo muy utilizado para propósitos similares a este en el que la comunicación se produce entre dispositivos punto a putno con un conector como elemento de transmisión se ha escogido el protocolo Modbus.

Modbus aplicado a comunicación serie [46] es un protocolo maestro-esclavo donde solo hay un maestro (en nuestro caso el módem) y uno o varios esclavos (contadores) conectados al mismo bus.

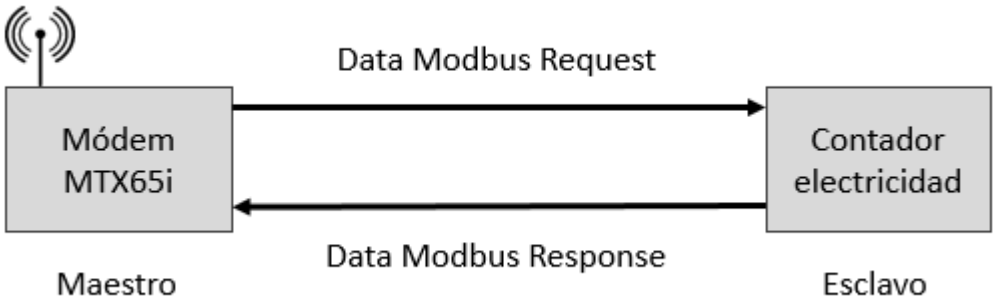

Ilustración 4-13. Esquema de comunicación y mensajes entre maestro y esclavo

Utilizaremos el modo unicast, en el que cada comunicación se compone de un mensaje request desde el maestro y otro response por parte del esclavo, que no podrá comunicarse sin un request previo por parte del esclavo.

La PDU quedará compuesta según estas características de la siguiente forma:

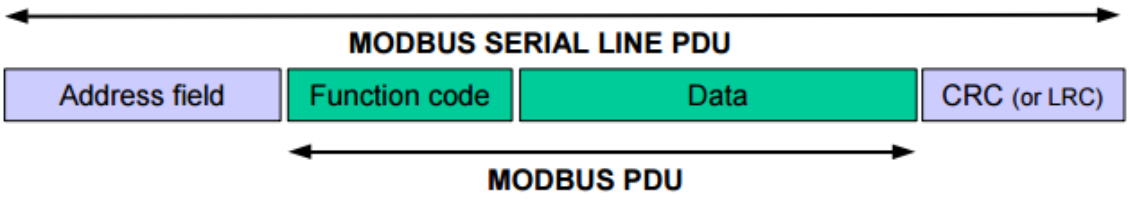

Ilustración 4-14. Formato de PDU en el protocolo Modbus aplicado a comunicación serie

El campo de dirección solo es necesario en el caso de los esclavos ya que en la otra dirección solo existe un maestro al que comunicarse. El código de función sirve para indicar al esclavo que acción quiere el maestro que realice (por ejemplo, envio de medidas).

Los bytes de datos se transmite primero el bit menos significativo. Estos datos pueden ser transmitidos utilizando el modo RTU (los campos son directamente un flujo de bits) o en modo ASCII en el que los bytes de datos representan caracteres según esta codificación, que será la utilizada:

Start	Address	Function	Data	LRC	End
1 char :	2 chars	2 chars	0 up to 2x252 char(s)	2 chars	2 chars CR,LF

Ilustración 4-15. Trama con caracteres ASCII

4.4.3.1 Formato de los datos recibidos por Modbus

Para el envío de los datos los mensajes que utilizamos en serán Request Data y Response Data cuyo contenido se muestra en la ilustración:

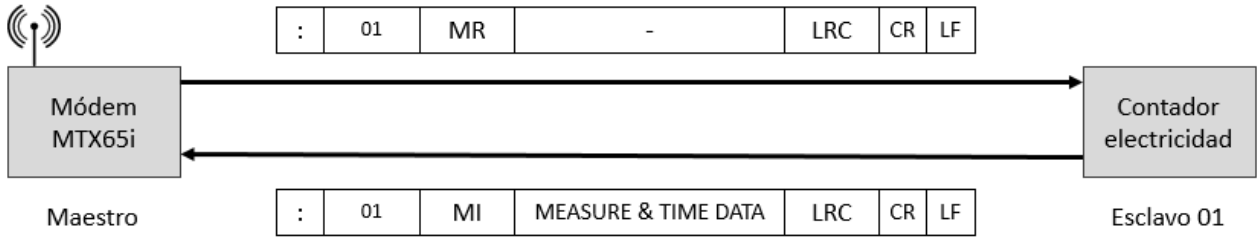

Ilustración 4-16. Tramas de datos ASCII entre módem y contador

En ambas tramas se indica el número de esclavo ya que en la del maestro se requiere la medida a es esclavo en concreto y en la respuesta indica de cuál de los posibles viene la información. El código de operación será en la trama del esclavo Measure Request (MR) y en la respuesta Measure Info (MI) indicando que devuelve la información solicitada.

4.5 Software a instalar

4.5.1 Firmware del Módem Siemens MTX65i

Este dispositivo cuenta con la versión de firmware 2.004 que incluye un cambio sustancial en cuanto a las anteriores. El cambio consiste en que el nombre de la ruta de directorios que contiene la librería de J2ME diseñada para el modem pasa de *com.siemens.** a *com.cinterion.**.

En cuanto al firmware se encuentra basado en la plataforma Java ME explicada en el apartado anterior JME.

El módem permite además que podamos comunicarnos a través de comandos Hayes (más popularmente conocidos como comandos AT) con los que podemos llevar a cabo cualquiera de las acciones de interacción con sí mismo y con la red que contiene. Una muestra de los mismos así como una relación de los más relevantes será incluida en el Anexo B.

4.5.2 Eclipse Helios SR2 versión 3.6.2 [47]

Se trata del entorno de desarrollo en el cual podremos programar la aplicación incluyendo las herramientas necesarias mediante la plataforma Eclipse Mobile cuyo paquete se denomina Pulsar y contiene lo necesario para el desarrollo. Aunque no se trata de las versiones más recientes de estas herramientas son las que el distribuidor recomienda utilizar para evitar problemas tanto en tiempo de ejecución como de compilación y de configuración en general de las mismas. También ofrece la posibilidad de utilizar NetBeans IDE versión 6.9.1.

Eclipse, además de tratarse de la plataforma más extendida para este dispositivo, cuenta con numerosas ventajas de sobra conocidas que aporta [48], como ser multiplataforma y contar con numerosos plugins disponibles para añadir multitud de utilidades.

Eclipse dispone de un editor de texto con analizador sintáctico. La compilación es en tiempo real, incluye desarrollo de pruebas con JUnit, control de versiones con CVS y wizards para asistir la creación de proyectos, clases, etc.

4.5.3 Java Development Kit (JDK) versión 1.6.0_25 [49]

Contiene las herramientas necesarias para desarrollar aplicaciones en java como son librerías de clases, compilador de Java, Webstart y archivos para applets, etc., además del Java Runtime Execution (JRE). El JRE es una versión reducida sin compilador que contiene la JVM y otros componentes para ejecutar aplicaciones y applets de java únicamente.

4.5.4 SDK del módulo TC65i

El CD de instalación Cinterion Mobility Toolkit (o en nuestro caso el Zip proporcionado por el proveedor oficial de Gemalto) contiene los siguientes elementos.

- Wireless Toolkit. El WTK es el directorio en el que se encuentran todos los componentes necesarios para la creación específica de aplicaciones Java ME y su depuración. La versión se encuentra en un archivo de texto en "Archivos de programa \ Cinterion \ CMTK \ <nombre del producto> \ WTK \ VersionWTK.txt".

- Doc con la documentación HTML

- Lib es el directorio donde se encuentra la librería J2ME con las clases necesarias almacenadas en un fichero denominado classes.zip

- Ejemplos WTK que se hallan en el directorio "All Users \ Cinterion ABC2 WTK Examples" que son ejemplos específicos de aplicaciones en JME (J2ME).

Module Exchange Suite. La configuración MES se encuentra "Installer \ MES Setup.exe". El MES proporciona herramientas para acceder al sistema de archivos Flash a través de una interfaz en serie desde el explorador de Windows. MES se puede instalar de manera independiente o como parte del CMTK como veremos en el apartado de instalación

Preparación del entorno.

- IMP debugging connection, para depurar la conexión. La configuración se encuentra en "Installer\IMPDbgConnectionSetup.exe". Además instala un módulo para la depuración del dispositivo. Igual que en el caso de MES se puede instalar de manera independiente o dentro del instalador del CMTK.

- SDK de Java para la programación de aplicaciones de java, es necesario tanto el IDE escogido (En nuestro caso Eclipse) como el Java Development Kit que contiene la JVM:

    - JDK (versión 6.25)

    - NetBeans IDE (versión 6.9.1 para 32 y 64 bits)

    - Eclipse Helios SR2 (versión 3.6.2 para 32 y 64 bits)

- Documentación con distintos ficheros de información del dispositivo (hardware, fuente de alimentación, etc.) así como anexos como por ejemplo una lista de comandos AT o una Guía Java para Usuarios en la que se desarrollan desde cero algunos ejemplos para ayudar en el aprendizaje y familiarización con el software [50].

4.5.5 Putty versión 0.64 [51]

Software con licencia libre que permite establecer conexiones Telnet, SSH, TCP o Serie a través de una interfaz similar al terminal de comandos. Lo utilizaremos para establecer conexiones en Windows 7 ya que en esta versión no disponemos del HyperTerminal.

El software está disponible para su descarga en internet en su página oficial. [26]

4.5.6 Minicom versión 2.7 [52]

Se trata de un software emulador de terminal que se utiliza en distribuciones Linux para establecer comunicaciones mediante el puerto serie a través de un terminal.

Cumple la función de permitirnos establecer una comunicación serie utilizando Linux.

4.5.7 Apache2 versión 2.4.7 [53]

Dentro de Apache incluiremos el servidor HTTP que intercambiará información con el módem a través de peticiones.

4.5.8 PHP versión 5.5.28 [54]

Versión de PHP que utilizaremos en el servidor Apache.

5 PREPARACIÓN DEL ENTORNO

5.1 Instalación del Software

Vamos a pasar a la instalación del SDK de Cinterion versión 2.004 para poder desarrollar aplicaciones MIDlet.

Para ello es necesario disponer de los archivos que vienen en el CD de instalación que el proveedor suministra o en caso de no tener dicho CD, ponerse en contacto con el proveedor para que lo envíe. Si esto ocurre el proveedor proporcionará un enlace al que podréis acceder mediante autenticación con usuario y contraseña. Desde ahí basta con descargar los ficheros del CD de instalación.

Para que tengamos una visión general de los principales pasos de la instalación tenemos este diagrama de secuencia:

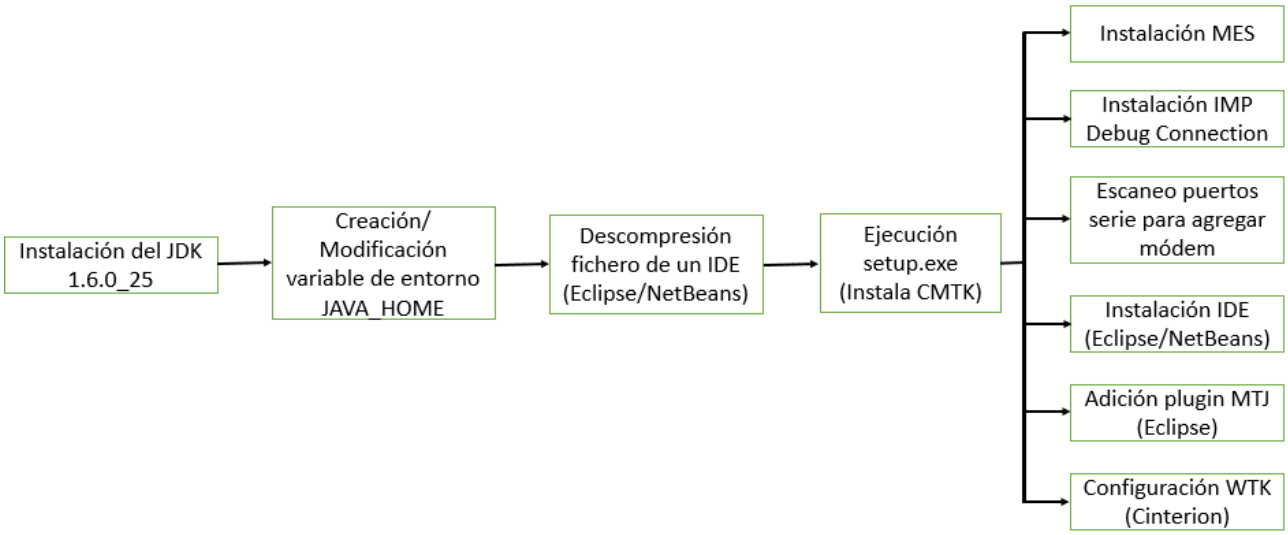

Ilustración 5-1. Diagrama de secuencia de la instalación del SDK

En él podemos ver como tras los preparativos para ejecutar el setup.exe éste archivo ejecuta dentro de su propia instalación la del resto de componentes necesarios como el MES para acceder a la memoria del dispositivo, el IMP Debug Connection, etc.

A continuación pasamos a explicar en detalle la instalación completa.

5.1.1 Requisitos del sistema

El kit de herramientas de Cinterion CMTK que se encuentra en el CD de instalación necesita que el equipo en el que se instale tenga:

- Windows XP, Windows Vista o Windows 7 instalado
- 110 MB de espacio libre en disco para la CMTK (sin JDK e IDE)

- Permisos de administrador
- Java Development Kit SE 6 Update 25.

5.1.2 Instalación de las herramientas de desarrollo [55]

Para instalar la versión JDK 1.6.0_25 mencionada en el apartado Requisitos del sistema es necesario ejecutar como administrador el fichero jdk-6u25-windows-i586.exe que se encuentra en la carpeta "Contribution" del CD de instalación. Ya solo es necesario seguir las instrucciones que aparecen en el instalador.

Una vez se ha instalado el JDK adecuado es necesario modificar la variable de entorno de Windows, en nuestro caso Windows 7 para que cuando utilicemos cualquier herramienta (como eclipse) que necesite usar por ejemplo la máquina virtual JVM o el compilador sepa donde se encuentra. Los pasos a seguir son los siguientes:

- Panel de Control
- Sistema
- Configuración avanzada del sistema
- Variables de entorno…
- Añadir si no se encuentra o modificar la variable JAVA_HOME cuyo valor será el directorio donde se haya instalado el JDK. En nuestro caso el valor será "C:\Program Files\Java\jdk1.6.0_25" [56].

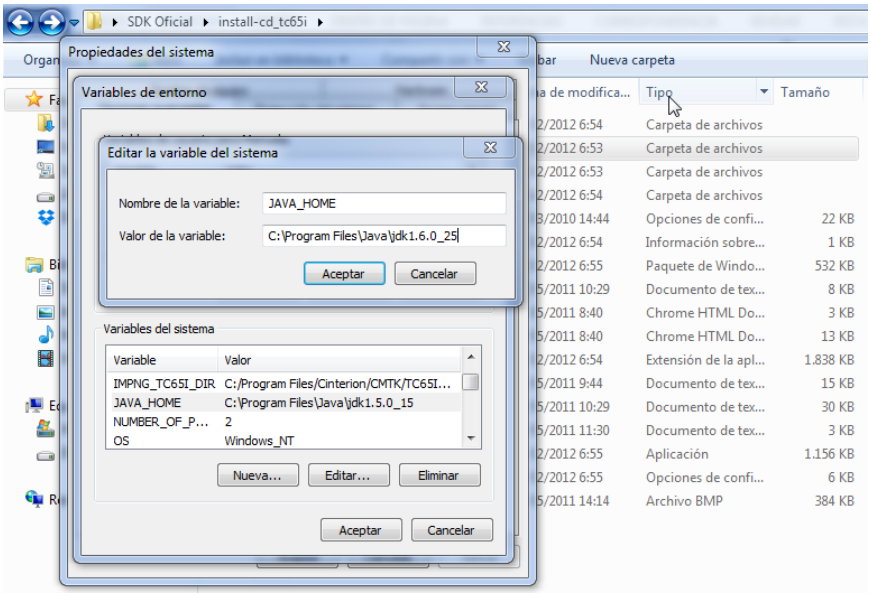

Ilustración 5-2. Modificación de la variable de entorno JAVA_HOME en Windows7

Necesitamos descomprimir uno de los IDEs incluidos en el CD de instalación para que durante la instalación de CMTK que viene a continuación se instale a la vez (si no tenemos ningún IDE instalado). No es se trata de una condición indispensable pero es lo que el proveedor recomienda porque contiene herramientas necesarias incorporadas y configuradas para el desarrollo de aplicaciones. Si instaláramos otra versión o esta misma de manera independiente al instalador de CMTK deberíamos instalar plugins adicionales para el desarrollo de aplicaciones MIDlet como el Mobile Tools for Java.

De los dos IDEs incluidos, en nuestro caso se ha elegido Eclipse por tratarse del más extendido y utilizado con vista a obtener una mayor información disponible en cuanto a los problemas que puedan surgir.

Descomprimimos el elegido en la carpeta “Contribution” en la que se encuentran los IDEs y ejecutamos el archivo de instalación del CMTK al completo denominado Setup.exe en el CD de instalación que se encargará de instalar todos los componentes (MES, IMP Debug, IDE, Mobile Tools for Java, WTK) durante un mismo proceso.

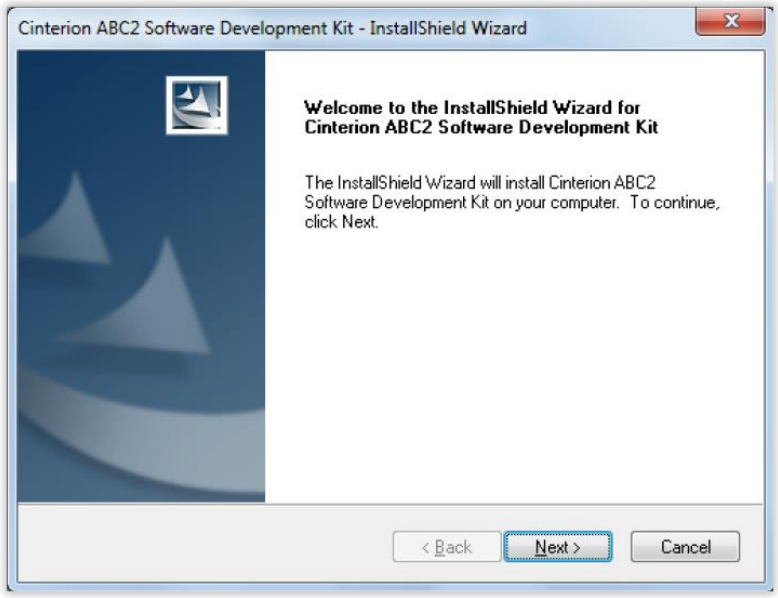

Ilustración 5-3. Instalador del CMTK de Cinterion

Es necesario seguir las instrucciones que el instalador nos vaya mostrando y clicar en “Siguiente” una vez hayamos concluido los requerimientos de cada paso.

En el paso que se muestra en la captura al darle a siguiente aparecerá el instalador del Module Exchange Suite que permite que accedamos a la memoria del dispositivo como si fuera un directorio del sistema al conectarnos por USB y cuya instalación concluiremos pulsando “Finish”.

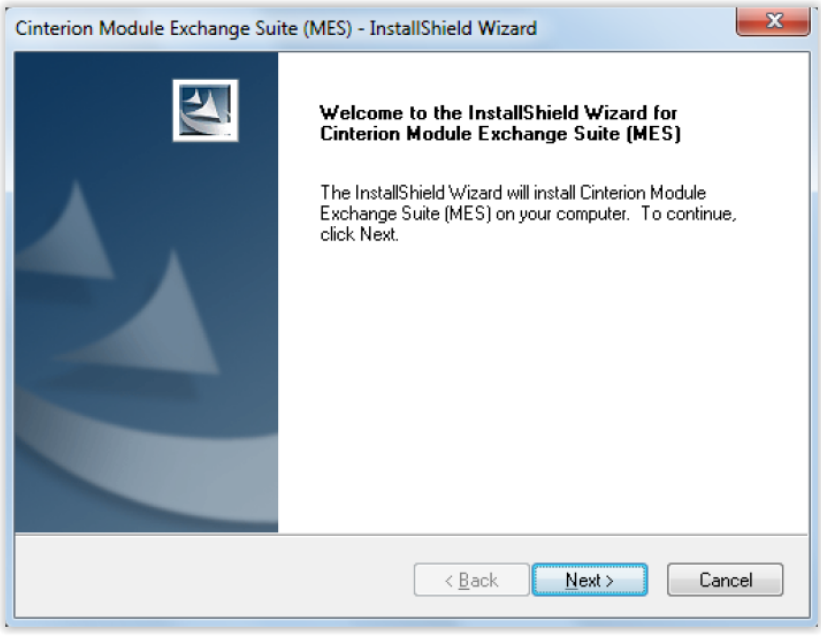

Ilustración 5-4. Captura de la ventana de instalación de MES

De igual manera al caso anterior aparecerá el instalador del IMP Debug Connection que nos permitirá depurar la conexión del puerto serie del módem al ordenador incluso en el caso de que utilicemos el conector USB a tal efecto.

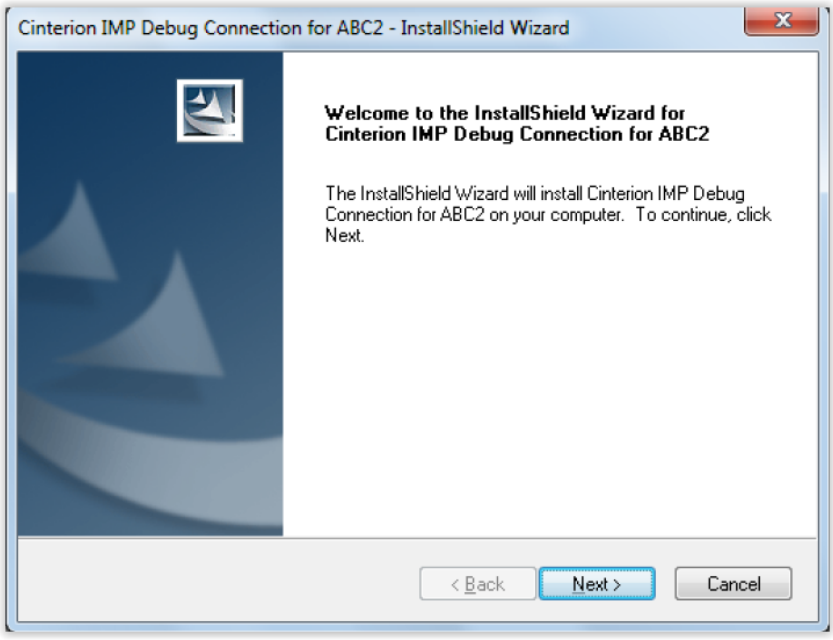

Ilustración 5-5. Captura de pantalla de la ventana de instalación de IMP Debug Connection

Aquí el instalador nos dará la opción de escanear los puertos serie en busca de dispositivos compatibles. Tenemos dos opciones: o posponerlo y escanearlos después con esta herramienta y el módem conectado por puerto serie, o conectar el módem en este momento y escanear para seleccionar el puerto correspondiente.

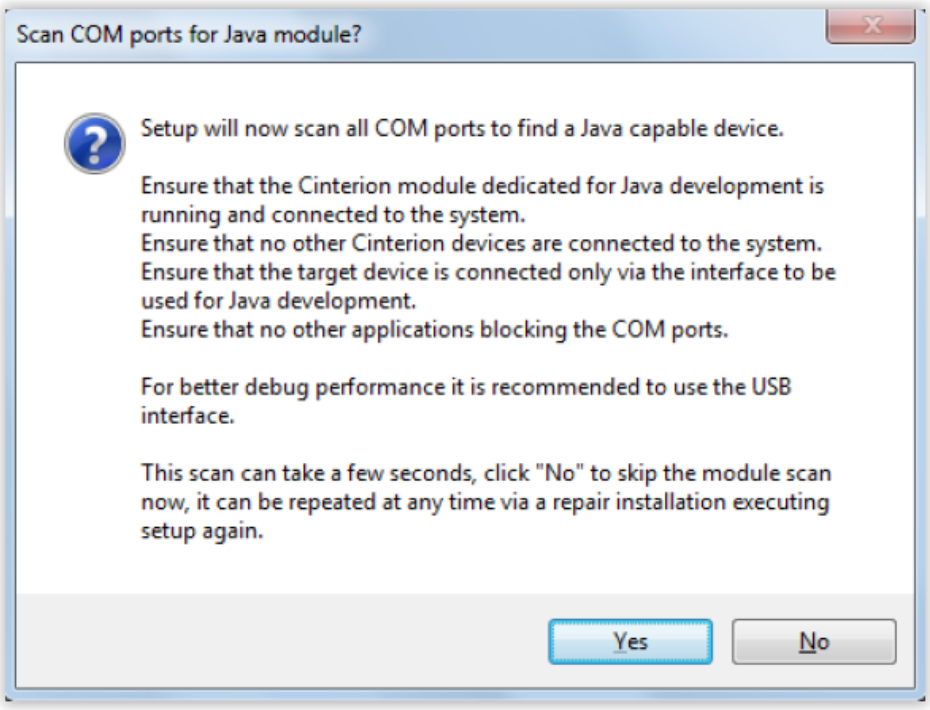

Ilustración 5-6. Captura de pantalla del Scan COM ports

Si elegimos escanear ahora el módem solo tenemos que conectar el cable Micro USB/USB al portátil y será detectado en uno de los puertos COM que aparecerá denominado "COMn" (COM con el número que corresponda). Una vez seleccionado pulsamos "Finish".

En el caso de saltar este paso, para poder escanear los puertos después es necesario ir al "Panel de Control", "Programas y características", y en "IMP Debug" elegir la opción "Cambiar".

El siguiente paso consiste en la instalación del IDE. Para ello en primer lugar se busca en el sistema algún IDE instalado previamente a través de la siguiente ventana:

Ilustración 5-7. Ventana de búsqueda de IDEs instalados en el sistema

Pueden darse dos situaciones: que tengamos un IDE instalado en el equipo, en cuyo caso será necesario que durante el escaneo el IDE se encuentre cerrado, o que por el contrario no tengamos ninguno. De darse esta situación el instalador ofrece la posibilidad de instalar el IDE de Eclipse Pulsar disponible en el CD de instalación (descomprimido previamente en el directorio en el que se encuentra).

Ilustración 5-8. Ventana de instalación del IDE Eclipse Pulsar

Si clicamos "Sí" el instalador se encargará tanto de instalar Eclipse como de completar la instalación de del CMTK de Cinterion pulsando "Finish".

Finalizando la instalación del CMTK tenemos incluido (si todo ha ido bien) el WTK pero para poder utilizarlo

cuando usemos Eclipse es necesario incluir un plugin adicional denominado Mobile Tools for Java (MTJ) [57].

Se trata de una extensión para eclipse que mediante un conjunto de herramientas y entornos permite el desarrollo de aplicaciones Java para dispositivos móviles y sistemas embebidos en general, de manera que su función es la de darnos las herramientas generales para poder utilizar las específicas del WTK.

La manera de añadir el plugin es abrir Eclipse (ya instalado). Al abrir Eclipse nos pedirá que indiquemos el workspace (se recomienda utilizar el que aparece por defecto para que la ruta al directorio no sea muy extensa). Una vez abierto hacer clic en "Help", "Install new software…".

En la ventana abierta hay que seleccionar la fuente de descarga que en este caso es "Helios - http: // download.

eclipse.org/releases/helios". Si clicamos "Add" aparecerán entre otros "Mobile and Device Development". De la lista desplegable tenemos que seleccionar Mobile Tools for Java, como aparece en la imagen.

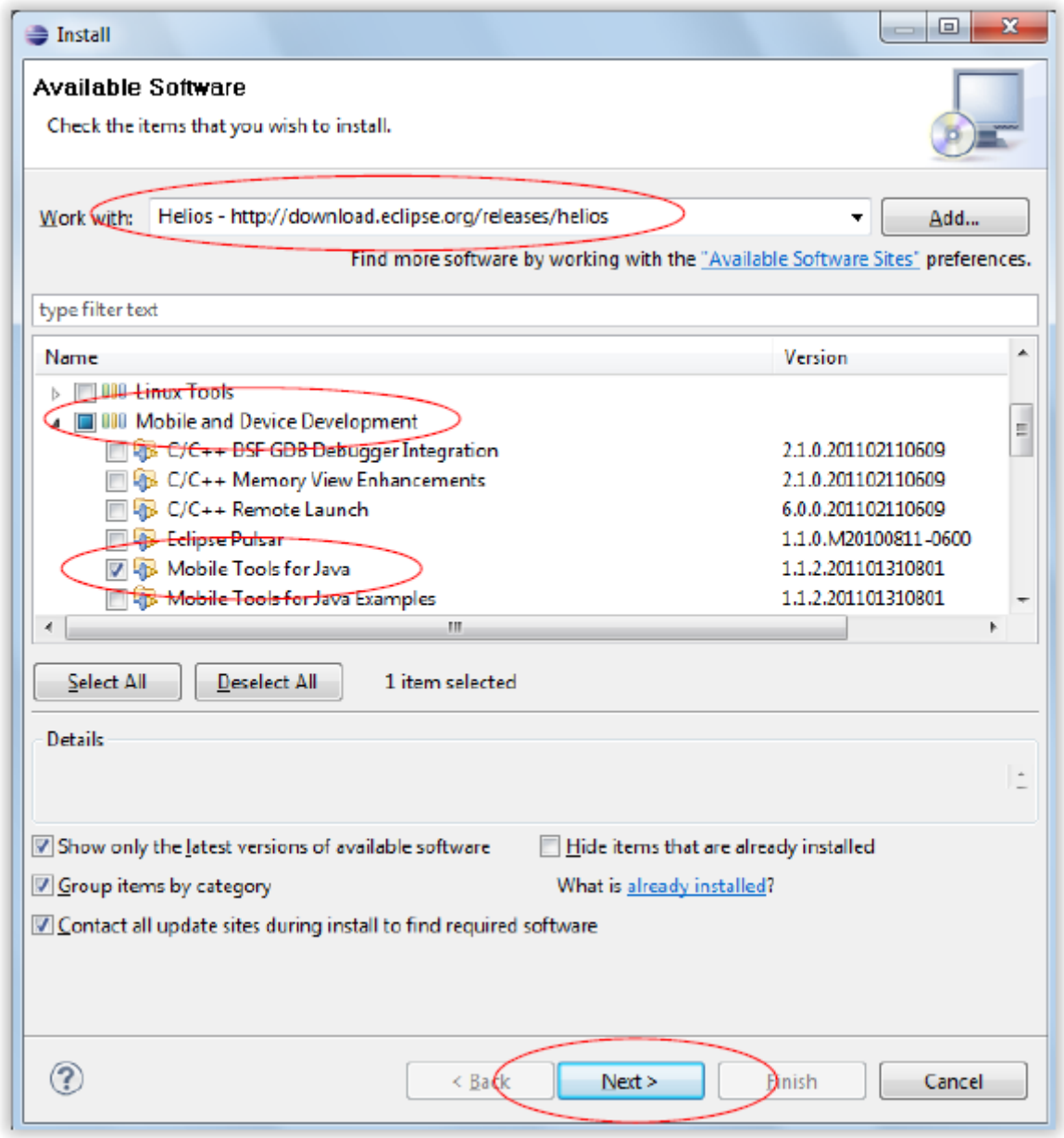

Ilustración 5-9. Ventana de instalación del plugin Mobile Tools for Java

A partir de aquí solo tenemos que seguir los pasos, hacer clic en “Next >” y posteriormente “Finish”.

Una vez tengamos el plugin MTJ tenemos que integrar el WTK en él.

Con Eclipse abierto vamos a la opción “Window”, posteriormente “Preferences”. Nos aparecerá una ventana con un listado a la izquierda del cual debemos seleccionar “Java ME” y dentro de éste “Manage Device” de forma que nos quedará la ventana como en la siguiente imagen:

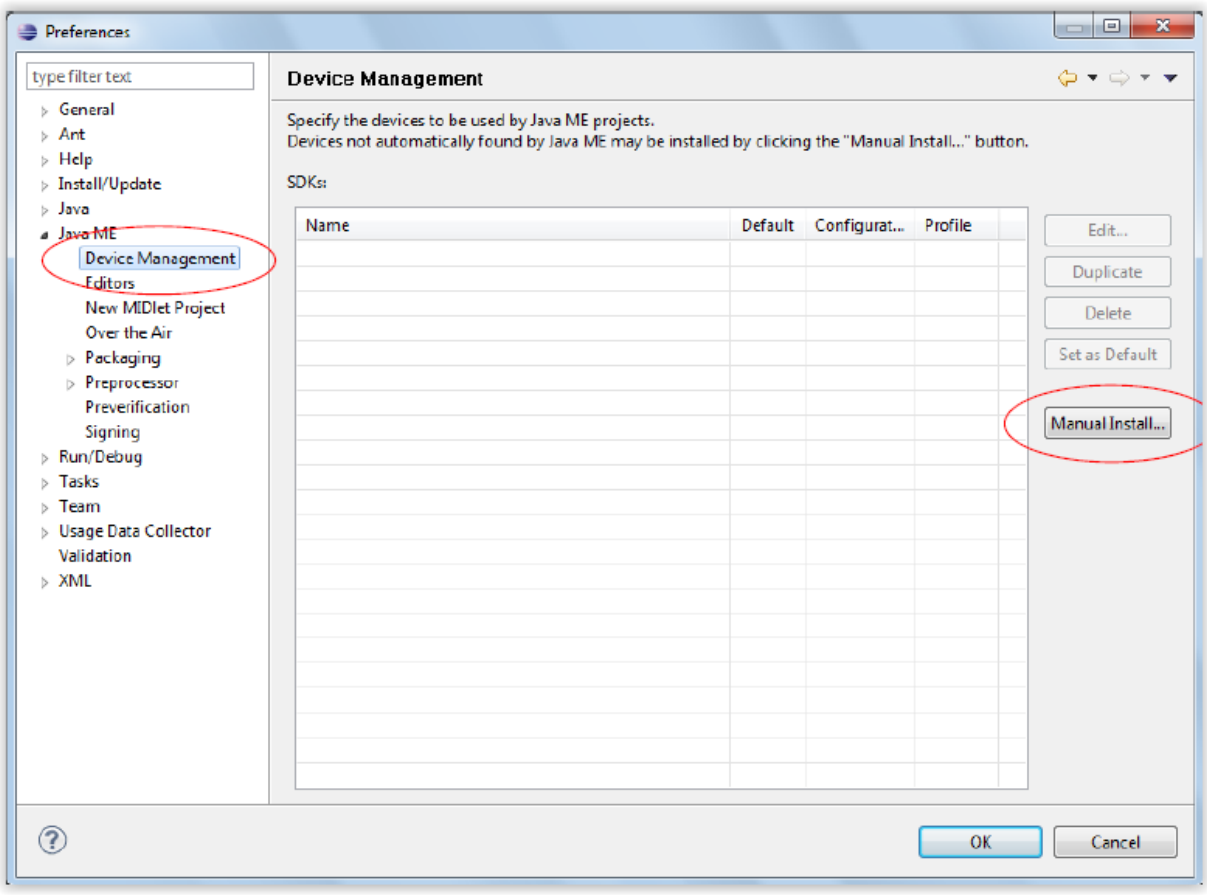

Ilustración 5-10. Captura de la ventana “Preferences” lista para incluir el dispositivo MTX65i

Como aparece en la imagen seleccionamos “Manual Install…” que nos abrirá otra ventana en la que tenemos que indicar el directorio donde se encuentra el archivo del WTK (en el CD de instalación de Cinterion):

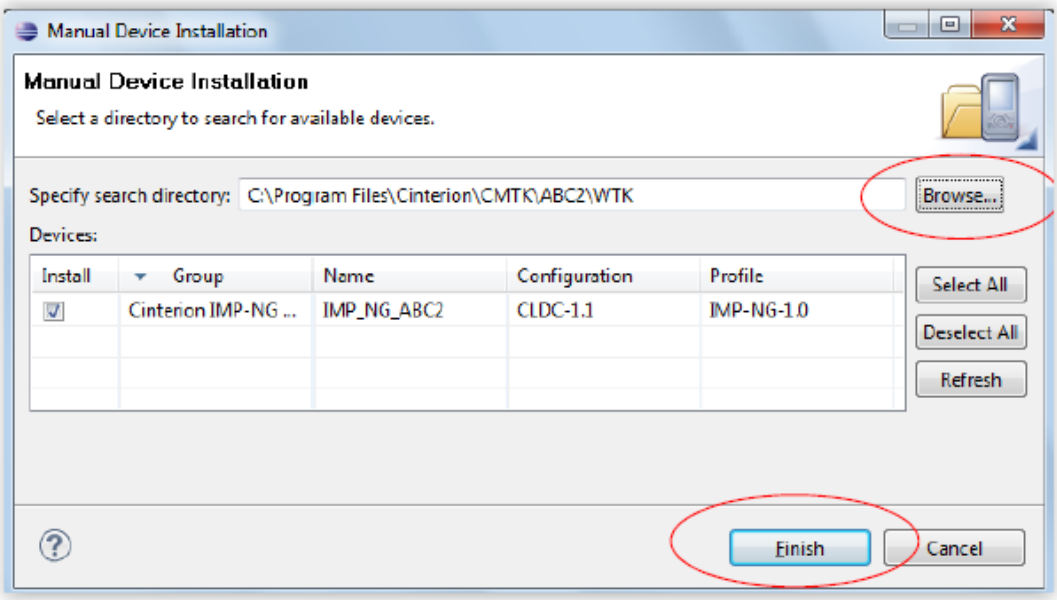

Ilustración 5-11. Captura de la ventana de Manual Device Installation

Una vez seleccionemos el directorio de WTK si pulsamos "Refresh" debe aparecernos el dispositivo tal y como ocurre en la figura lo que seleccionándolo y pulsando "Finish" quedará añadido.

Para configurar sus propiedades lo seleccionamos en la ventana "Preferences" y clicamos Editar.

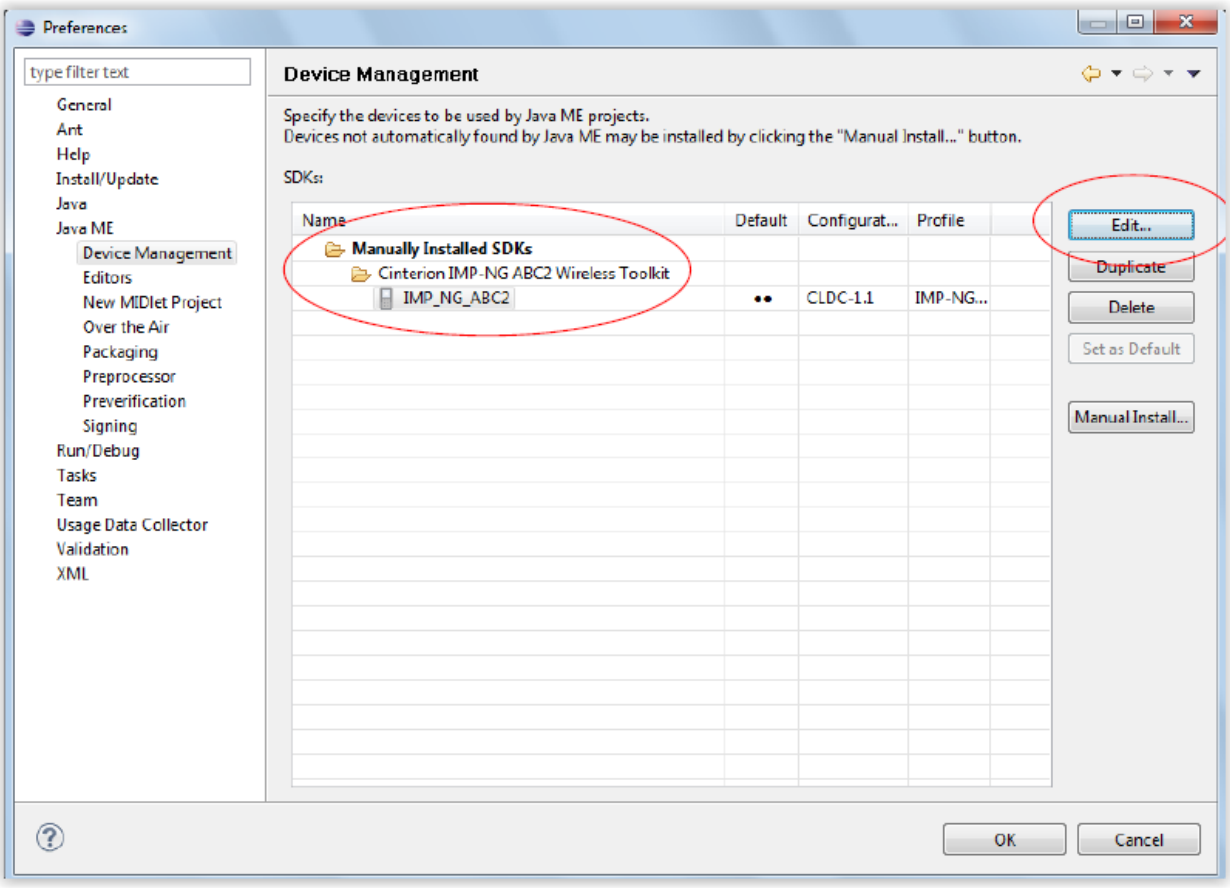

Ilustración 5-12. Captura de la ventana “Preferences” con las herramientas del dispositivo añadidas

Es necesario editar estas herramientas porque aunque por defecto se instala WTK hasta que no se incluye el plugin MTJ no podemos configurar todo para que nos aparezcan las clases soportadas por el módem o el enlace al Javadoc (que se encuentran en el directorio del CD de instalación).

En la siguiente figura se muestra como incluir la ruta del Javadoc:

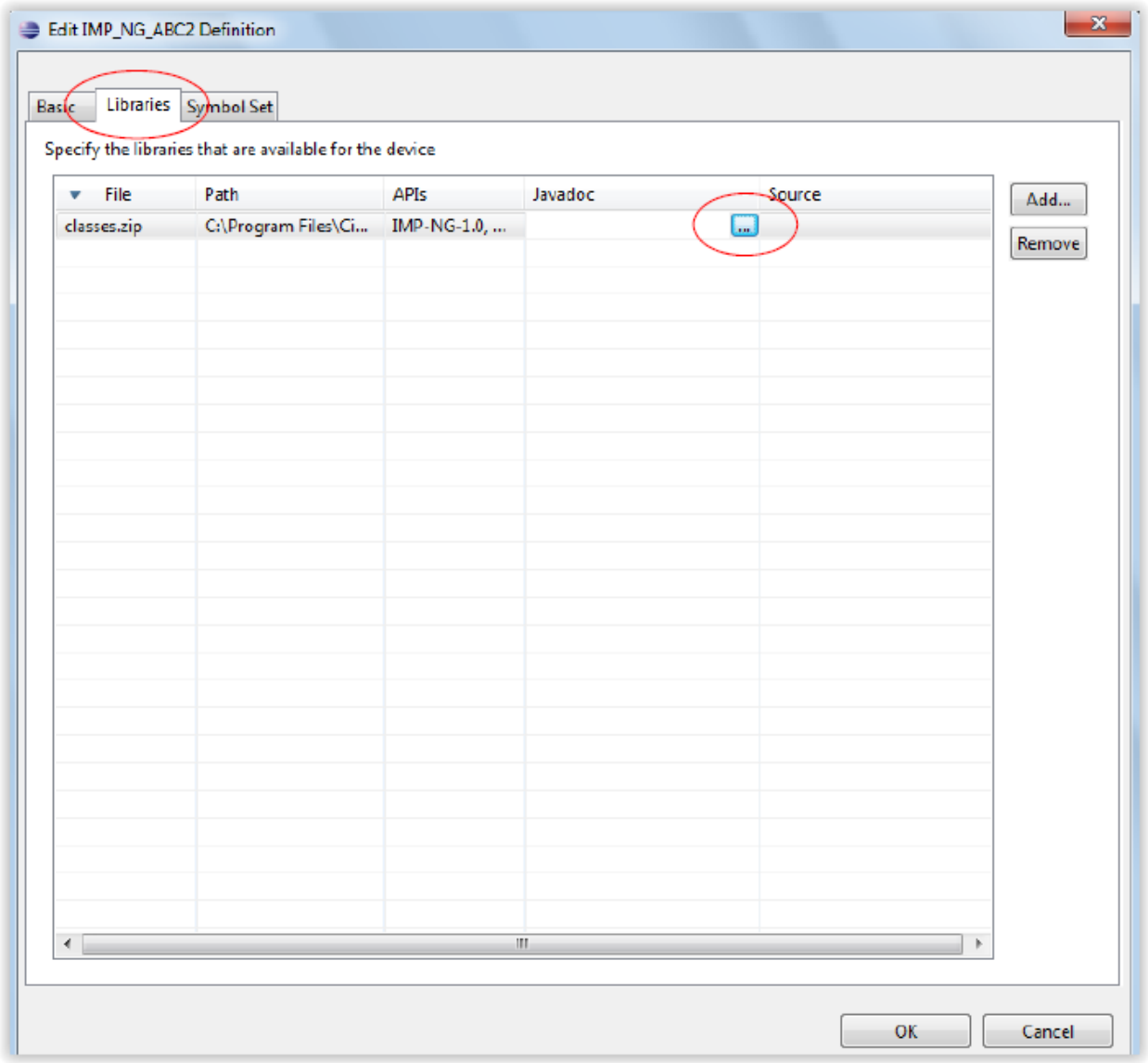

Ilustración 5-13. Captura de la ventana "Edit IMP-NG"

Con este último paso tendremos el SDK de Cinterion completamente instalado incluyendo el IDE elegido (Eclipse) con el plugin para desarrollo de aplicaciones para dispositivos móviles Mobile Tools for Java y el conjunto de herramientas específico para el módem MTX65i Wireless Toolkit que nos proporcionará las librerías con las clases que soporta este módulo, quedando el entorno completamente configurado para iniciar el desarrollo de aplicaciones como se describirá en el apartado Introducción a la programación de aplicaciones MIDlet.

Toda la información sobre la instalación del SDK ha sido extraída del manual destinado a tal efecto que incluye [58].

5.1.3 Problemas encontrados en la instalacion de las herramientas de desarollo

En nuestro caso, debido a la falta del CD se solicitó el SDK al distribuidor (Gemalto propietaria de Cinterion [59]), pero por tratarse del periodo estival, no recibimos el SDK hasta pasado un mes durante el cual se trabajó con una versión anterior (versión 1.100) que ha propiciado diversos problemas a lo largo del proceso de instalación.

- En primer lugar, dado ya contábamos con Eclipse instalado en su versión 4.3.2 pasamos a la instalación de un plugin adicional para adquirir las herramientas de Eclipse Mobile denominado EclipseME en su versión 1.6.8 sin el cual no podríamos programar aplicaciones MIDlet.

  El problema consistía en la incompatibilidad de versiones dado que el plugin estaba preparado para funcionar en la versión 3.2.2 de Eclipse. La solución a este problema fue la reinstalación de Eclipse en la versión requerida.

- De igual modo ocurría con la versión del JDK por lo que se reinstaló el paquete completo de Java.
- Al cambiar el JDK la máquina virtual de había dejado de funcionar lo que impedía el arranque de eclipse generando un error del tipo "*An error ocurred. See the log file C:\Users\Usuario\.eclipse\configuration\XXXXXXXXXX.log file*".

  Esto era debido a que no se modificó automáticamente el valor del directorio del JDK en la variable JAVA_HOME, que fue la solución al problema.

- Por último, utilizando la librería proporcionada por este SDK anterior y de nuevo propiciado por una incompatibilidad de versiones no se detectaban en tiempo de ejecución clases que sí existían en tiempo de compilación (no solo eran importadas sino que Eclipse compilaba el código sin problema). En este caso no era posible solucionar el error sin el SDK de la nueva versión proporcionado por el fabricante ya que, justo en la versión 2.004 que contenía el firmware del módem se produjo una modificación del nombre de la estructura de directorios, en la cual Cinterion decidió cambiar la nomenclatura de los mismos de *com.siemens.** a *com.cinterion.**. [60] A pesar de que las clases no cambiaban en esta nueva librería, el cambio de nombre propiciaba que al compilar no apareciesen errores pero al ejecutar con una versión de firmware con la nueva estructura se produjera el error NoClassDefFoundError. La solución pasa por utilizar el SDK de Cinterion en la última versión disponible.

5.1.4 Instalación de Putty [51]

Como indicamos en el apartado de Software a instalar si estamos utilizando el sistema operativo Windows 7 necesitamos un software que nos permita establecer conexiones (en nuestro caso serial) ya que en esta versión del SO no disponemos de HyperTerminal. Para poder utilizar putty solo necesitamos descargarlo de la página oficial que se encuentra aquí [51].

5.1.5 Instalación de Minicom [52]

La instalación del paquete de Minicom se puede hacer en Ubuntu desde los repositorios ejecutando el siguiente comando:

```
sudo apt-get install minicom
```

5.2 Primeros pasos con el módem MTX65i

Vamos a describir el procedimiento para arrancar por primera vez el módem y establecer la configuración que nos interesa tanto en el módem como en el ordenador al que lo conectemos.

5.2.1 Conexión y arranque

Para comenzar, introduciremos la SIM en el módem en el lugar indicado a tal efecto. Aunque no es imprescindible para una primera configuración del mismo, esto nos evitará tener que reiniciar el módem cuando nos dispongamos a fijar los parámetros de acceso a la red.

A continuación es necesario alimentar el módem con el cable proporcionado por el proveedor. En este momento el módem durante unos segundos estará en fase de arranque en la cual no podemos realizar ninguna acción sobre el mismo. Una vez alimentado se inicia de manera automática y puesto que no dispone de un botón de reset, la manera de reiniciar es desconectarlo de la corriente.

Una vez ha arrancado podemos conectarnos al módem de tres maneras.

5.2.1.1 Vía Micro USB/USB para acceder a la memoria

Este mecanismo será el adecuado cuando queramos introducir las aplicaciones MIDlet que hayamos programado en la memoria del módem para que se ejecuten cuando lancemos los comandos adecuados.

Si es la primera vez que nos conectamos por USB al módem será necesario configurar el Module Exchange Suite anteriormente instalado para tener acceso a la memoria del dispositivo desde el explorador de Windows.

Debemos tener en cuenta que para que esto ocurra el módem debe estar conectado y cada vez que accedamos mediante el MES los parámetros de configuración relativos al Baud rate se fijarán a la tasa que necesita esta herramienta que es una de las disponibles para el módem [61] como por ejemplo 115200. Tampoco debemos olvidar que el número máximo de objetos que podemos poner en dicha memoria es de 200 incluyendo archivos y subdirectorios.

Dicho ellos para configurar el acceso debemos ir a "Equipo". Debe aparecernos un icono como el de la siguiente imagen tras la instalación del CMTK:

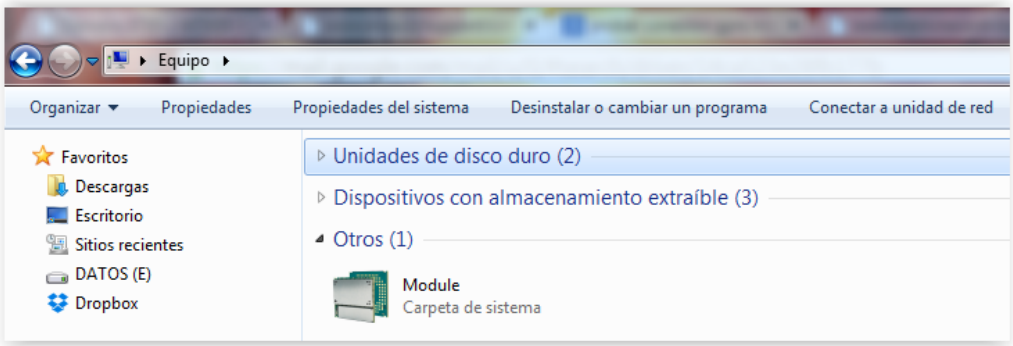

Ilustración 5-14. Captura de pantalla de la ventana "Equipo"

Hacemos click derecho en "Module" y escogemos "Propiedades". Debe aparecernos una ventana como esta:

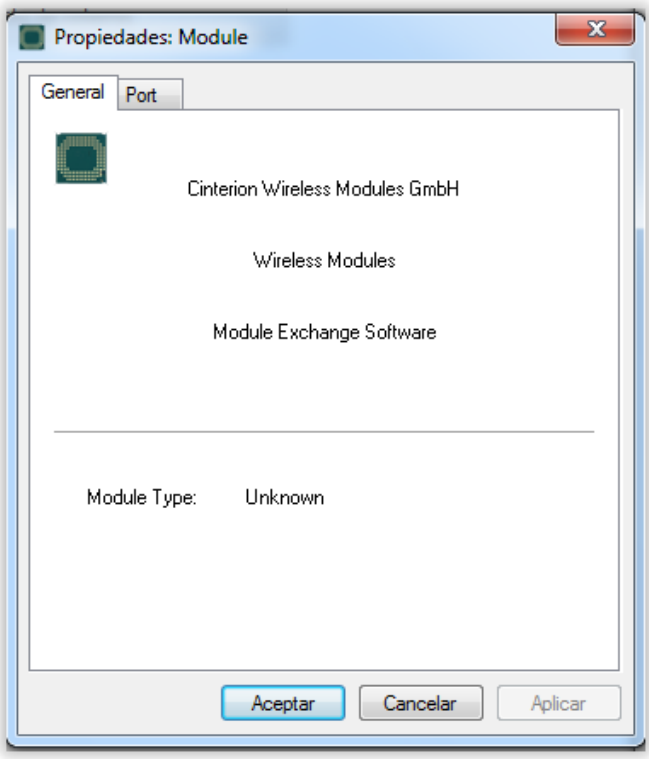

Ilustración 5-15. Captura de pantalla de “Propiedades” de “Module”

Vamos a la pestaña “Port” para escoger el puerto a través del cual queremos acceder a la memoria con lo que aparecerá:

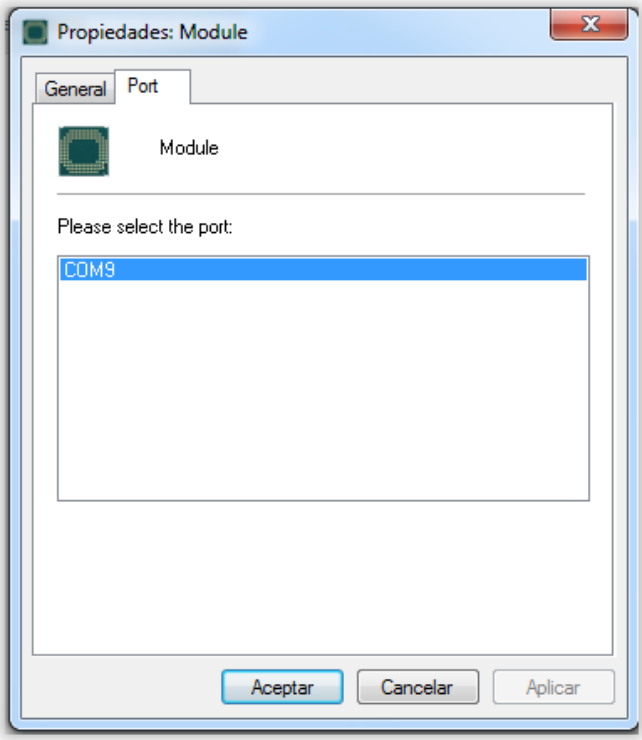

Ilustración 5-16. Captura de pantalla de la Selección de puerto

Seleccionamos el puerto que aparece, clicamos en "Aplicar", posteriormente a "Aceptar". Si hacemos doble clic en Module tendremos acceso a "Module Disk (A:) para poder incluir los ficheros de aplicación que desarrollaremos más adelante.

5.2.1.2 Vía Micro USB/USB para establecer una comunicación serie

Otra de las posibilidades es conectar igualmente el USB como en el caso anterior pero en lugar de acceder a la memoria del dispositivo será establecer una conexión gracias al cliente Putty instalado en el apartado Instalación de Putty . Haciendo doble clic sobre el archivo descargado durante la instalación nos aparecerá una ventana como la de la siguiente ilustración en la que podremos indicar los parámetros de la conexión que queremos establecer:

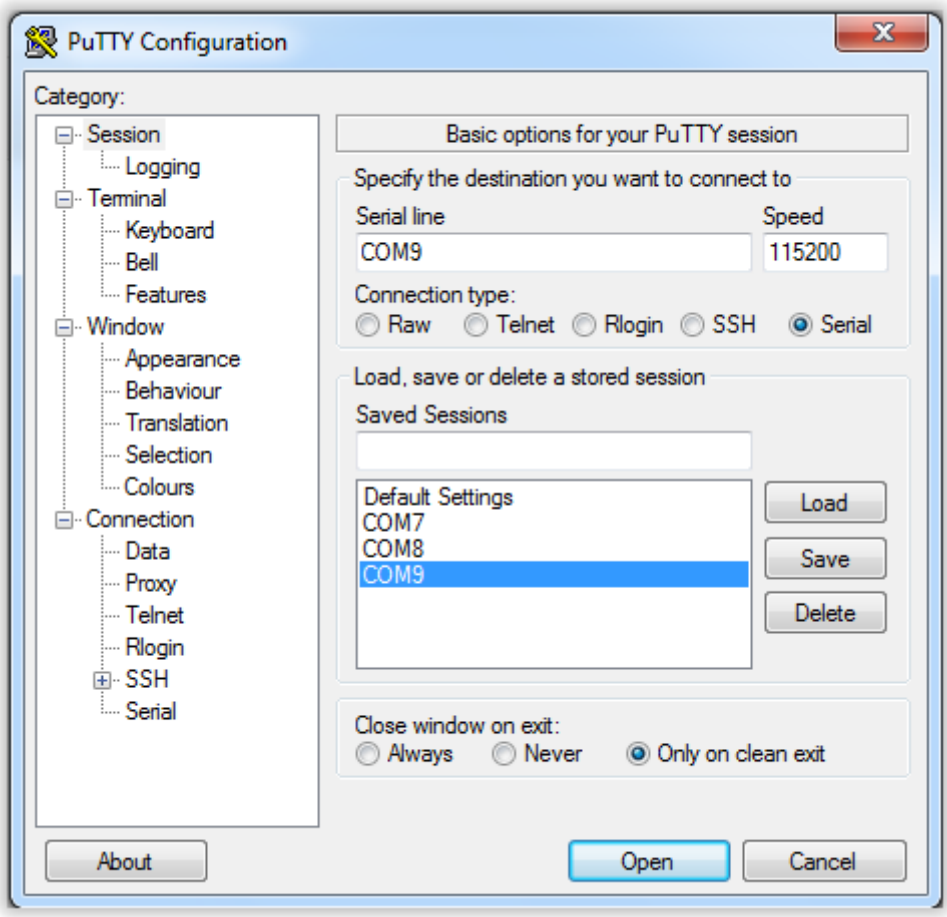

Ilustración 5-17. Captura de pantalla de la interfaz de Putty

Habrá que elegir el "COM" que se ajuste a nuestra conexión (lo podemos ver en las propiedades de Module) el baud rate en speed teniendo en cuenta que se trate de una de las permitidas por el fabricante [61]. Por último tampoco podemos olvidar elegir la opción "Serial" que es el tipo de comunicación que queremos establecer.

Con esto podemos iniciar la conexión clicando "Open". El aspecto de la interfaz así como algunos comandos de prueba aparecen en la siguiente imagen.

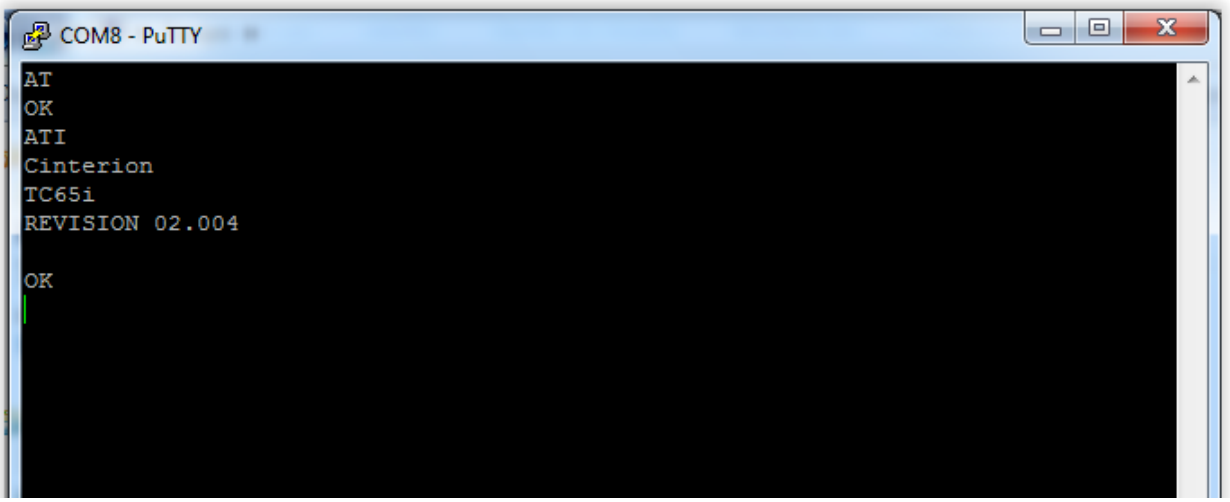

Ilustración 5-18. Captura de la interfaz de Putty con comunicación serie abierta con el módem

5.2.1.3 Vía puerto serie con un conector DB9 para establecer una comunicación serie

Si queremos utilizar un puerto serie real y no emulado como en el caso anterior debemos utilizar un equipo que cuente con este conector (DB9). En nuestro caso se trata de un ordenador de sobremesa que tiene instalado Ubuntu 14.04.

Para poder enviar datos en una comunicación serie necesitamos instalar en Ubuntu el paquete de Minicom como se indica en el apartado Instalación de Minicom .

Al conectar el módem mediante el cable DB9 correspondiente al ordenador tenemos que abrir una terminal y ejecutar el siguiente comando:

```
dmesg | grep tty
```

Así obtenemos la lista de mensajes del búfer del kernel [62] que es donde se genera la información relativa a la conexión del puerto serie como el nombre que se le asigna para poder utilizarlo.

La salida de este comando nos aportará el nombre asignado a nuestro puerto, en nuestro caso "/dev/ttyS0".

Solo falta ejecutar minicom y ajustar los parámetros de la conexión según nos indica el propio Minicom, por ejemplo 8n1 (8 bits de datos, ningún bit de paridad y 1 bit de parada), el baud rate 4800 y el nombre del puerto /dev/ttyS0. Con esta información ya podemos ejecutar:

```
sudo minicom -D /dev/ttyS0
```

Ahora lo que tecleemos en el terminal será enviado al módem y las respuestas del módem a los comandos insertados se mostrarán por pantalla (con el eco activado si hemos ejecutado el comando ATE1), quedando lo que se muestra en la siguiente figura:

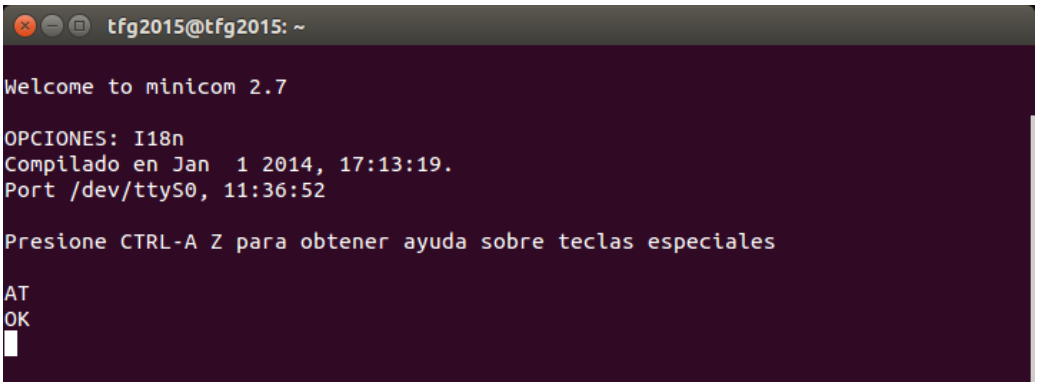

Ilustración 5-19. Captura de pantalla de la sesión establecida en la interfaz de Minicom

Puede ocurrir que en la configuración del módem no se encuentre especificado que queremos que la salida estándar sea por el ASC0. Un indicador de que esto ocurre es que vemos los comandos que mandamos al módem pero no vemos el OK en respuesta o similar. Para solucionarlo tenemos que ejecutar el comando:

AT^SCFG=”Userware/Stdout”,”ASC0” (o ASC1 si estamos conectados al segundo puerto del módem). Si cambiamos a utilizar el USB desde Windows como en el apartado anterior solo tenemos que sustituir en el comando “ASC0” por “USB”.

5.2.2 Configuración de la conectividad GPRS del módem

Uno de los pilares fundamentales para lograr el objetivo de la comunicación de datos es configurar el acceso a la red de datos del operador que hayamos elegido. Para este punto no olvidemos que la SIM tiene que estar insertada en el dispositivo.

El módem dispone de una serie de comandos que nos permiten realizar esta configuración si nos encontramos en el modo de comandos y no de datos. Pero si necesitamos configurar estos parámetros del perfil de conexión para que nuestra aplicación se conecte a un APN concreto podemos indicar la configuración de dicho perfil mediante el código de la aplicación, como veremos en el apartado de desarrollo de la misma Comunicación Módem – Servidor Central.

Centrándonos en el modo comandos los pasos a realizar son los siguientes.

Si comenzamos a utilizar comandos AT es una buena práctica activar la salida de códigos de error en formato alfanumérico para obtener toda la información posible de la siguiente manera:

AT+CMEE=2

Con esto quedará activado para que nos muestre por la salida estándar que tengamos establecida en AT^SCFG=”Userware/Stdout” (comando de configuración extendida) los mensajes de error completos.

En primer lugar es necesario conocer los datos de nuestro punto de acceso (nombre del APN, usuario y contraseña). En caso de que tengamos intención de conectarlos a través del APN de la compañía operadora estos datos estarán disponibles en Internet. En nuestro caso al tratarse de un APN perteneciente a una empresa se sustituirán los mismos por los del operador de Movistar, a modo de ejemplo.

Comprobamos que nuestra tarjeta no tiene código PIN mediante el comando en modo consulta:

AT+CPIN?

Si no está protegida la respuesta a este comando será:

+CPIN: READY

En caso contrario podemos obtener:+CPIN: SIM PIN

Lo que significará que se encuentra a la espera de que se le introduzca el PIN, por lo que tendremos que ejecutar el comando en modo escritura:

AT+CPIN = 1234

Donde 1234 debe ser sustituido por el código PIN correspondiente. Cuando introduzcamos el código correcto debe aparecer +CPIN: READY

Ahora comprobamos que el módem se encuentra en el modo de funcionamiento completo (puede encontrarse en algún modo "sleep" en cuyo caso no podríamos realizar la conexión). Para ello ejecutamos:

AT+CFUN

Debe devolvernos 1. Si no es así lo activaremos nosotros mismos ejecutando el comando en modo escritura:

AT+CFUN=1

Si ejecutamos el comando AT+COPS en modo consulta y lectura veremos los operadores disponibles y si actualmente está ligado a algún operador respectivamente.

En nuestro caso:

```
AT+COPS=?
+COPS: (2,"vodafone ES",,"21401"),(1,"movistar",,"21407"),(1,"Orange",,"21403"))

OK
AT+COPS?
+COPS: 0,0,"vodafone ES"
```

Ilustración 5-20. Captura de la ejecución del comando AT+COPS en modo consulta y lectura

Vemos la lista de operadores disponibles (incluido el operador actual marcado con un 2).

Además si lo ejecutamos en modo lectura vemos que la selección de operador está en modo automático (primer parámetro devuelve 0). También vemos que se está usando el formato alfanumérico para mostrarnos el operador (segundo parámetro a 0), y finalmente nos muestra el operador actual ("Vodafone ES").

Si automáticamente no se conecta al APN de ningún operador podemos ejecutar el comando AT+COPN en modo ejecución para ver los operadores disponibles:

```
+COPN: "73002","Movistar"
+COPN: "73003","CLARO CHL"
+COPN: "73010","CL ENTEL PCS"
+COPN: "732101","COMCEL"
+COPN: "732103","COL MOV / TIGO"
+COPN: "732111","COL MOV / TIGO"
+COPN: "732123","COL Movistar"
+COPN: "73401","DIGITEL GSM"
+COPN: "73402","DIGITEL GSM"
+COPN: "73403","DIGITEL GSM"
+COPN: "73404","movistar"
+COPN: "73406","VE MOVILNET"
+COPN: "73601","VIVA"
+COPN: "73602","BOMOV"
+COPN: "73603","Telecel Bolivia"
+COPN: "73801","DIGICEL"
+COPN: "738002","GUY CLNK PLS"
+COPN: "74000","movistar"
+COPN: "74001","PORTA GSM"
+COPN: "74401","HOLA PARAGUAY S."
+COPN: "74402","CLARO PY"
+COPN: "74404","Telecel GSM"
+COPN: "74405","PY Personal"
+COPN: "74602","SR.TELESUR.GSM"
+COPN: "74603","DIGICEL"
+COPN: "74604","UNIQA"
+COPN: "74807","MOVISTAR"
+COPN: "74810","CLARO URUGUAY"
+COPN: "750001","C&W FLK"
+COPN: "79502","TM CELL"
+COPN: "90112","MCP Maritime Com"
+COPN: "90114","AeroMobile"
+COPN: "90115","OnAir"
+COPN: "90117","Navitas"
+COPN: "90118","WMS"
+COPN: "90121","Seanet"
OK
```

Ilustración 5-21. Captura de la ejecución del comando AT+COPN

En el caso de querer conectarnos a uno de los elementos devueltos por este comando tenemos que ejecutar el mismo comando pero en modo escritura:

AT+COPN = 1,0,”opnameInNumericFormat”

En esta ocasión no podemos utilizar caracteres alfanuméricos para el nombre del operador, sino que debemos incluirlo con el MCC (Mobile Country Code) y MNC (Mobile Network Code) [63] correspondiente como aparece en la lista (“90121” por ejemplo).

Para terminar con la selección del operador, si se da la circunstancia de que nuestro operador no aparece en la lista debemos insertarlo nosotros mismos mediante el comando AT+SRPN con la siguiente sintaxis:

AT^SRPN=1,"20404","MiVF","Mi Vodafone"

Donde el primer parámetro a 1 se utiliza para indicar que queremos crear una entrada en la lista de operadores disponibles, “20404” sería el MCC MNC anteriormente citados y las cadenas “MiVF” y “Mi Vodafone” serían el nombre corto y completo respectivamente del operador que tenemos intención de incluir.

Las opciones que incluyen selección manual están particularmente desaconsejadas, no solo porque el dispositivo cuenta con la capacidad de conectarse según la SIM introducida a la más apropiada, sino también porque la selección manual de un operador no disponible puede conllevar cambios en parámetros de configuración difíciles de detectar y causar problemas como los que se especifican en el apartado Problemas en la instalación y configuración del módem.

Ahora creamos un contexto PDP. Con él especificamos el APN y el tipo de paquetes que se enviarán:

AT+CGDCONT=1,"IP","movistar.es"

Para vincularnos al nodo SGSN del operador al que estamos conectados ejecutamos:

AT+CGATT=1

El siguiente paso es crear el perfil para la conexión GPRS, que se realiza con los siguientes comandos:

AT^SICS=1, conType, GPRS0

Selecciona el tipo de conexión (GPRS0 para GPRS en nuestro caso, o CSD Circuit-switched data call).

AT^SICS=1, inactTO,"0"

Define el timeout por inactividad (0 indica que no se utilizará timeout).

AT^SICS=1, dns1,"193.254.160.1"

Opcional, para fijar un servidor DNS primario.

AT^SICS=1, authMode,"PAP"

Obligatorio para fijar el protocol de autenticación.

AT^SICS=1, apn,"movistar.es"

Opcional para incluir el nombre del APN.

AT^SICS=1, “user”,”MOVISTAR”

Usuario opcional para el acceso al APN.

AT^SICS=1, “passwd”, “MOVISTAR”

Contraseña opcional para el acceso al APN.

AT^SICS: 1,"alphabet","0"

Opcional para indicar el conjunto de caracteres que vamos a utilizar.

El 1 que incluimos como parámetro en todos los comandos AT^SICS es el identificador de perfil (en este caso será el perfil 1).

Una vez definido el perfil lo activamos mediante el comando:

AT^SICO=1

Donde 1 es el identificador del perfil.

Con esta secuencia quedaría lista la conexión GPRS mediante comandos AT. Durante el desarrollo de la aplicación veremos que indicando en el código las características del perfil de conexión (APN, user, password, etc.) podemos enviar datos vía conexión GPRS sin necesidad de realizar la secuencia de comandos descrita.

Si queremos validar que se ha realizado la conexión GPRS correctamente [64], ejecutamos el comando de estado de registro en la red en modo lectura:

AT+CGREG?

Que nos tiene que dar como resultado:

+CGREG=0,5

El 0 indica que los códigos URC están desactivados en la red y el 5 es que el dispositivo se encuentra enlazado a la red y disponible.

Además de los comandos AT utilizados para esta configuración, todos los relacionados con GPRS se pueden consultar en la siguiente referencia [65], en la Relación de Comandos AT relevantes y el manual de comandos AT disponible en el SDK de Cinterion.

5.3 Problemas en la instalación y configuración del módem

A los problemas surgidos durante la Problemas encontrados en la instalacion de las herramientas de desarollo, hay que sumar los que se han producido durante la puesta en funcionamiento del módem MTX65i cuya resolución se detalla a continuación con objeto de que sirvan de referencia y guía en caso de producirse.

- Para el acceso a la memoria flash mediante MES es necesario no olvidar la selección del puerto COM correspondiente, y si cambiamos a otro conector USB ajustarlo de nuevo ya que el nombre asignado al puerto varía de uno a otro. En ese caso solo hay que seguir los pasos detallados en Vía Micro USB/USB para acceder a la memoria.

- En la fase de Conexión y arranque puede ser que si no se establece correctamente la tasa de símbolos de la conexión por ejemplo Vía Micro USB/USB para establecer una comunicación serie, aparezcan en la interfaz caracteres ininteligibles o no aparezca nada.

  Esto significa que tenemos que ajustar el baud rate correctamente ya estemos utilizando Putty versión 0.64, o Minicom versión 2.7 .

- Por último en el caso de la Configuración de la conectividad GPRS del módem es muy importante realizar la elección de operador si es posible de manera automática dado que si se realiza de manera manual aunque no se complete con éxito, ciertos parámetros como por ejemplo la banda de frecuencia que utiliza el operador intentan adaptarse al nuevo operador, quedando reconfigurados.

  Esta nueva configuración hace que el terminal quede sin conexión ya que la señal que se recibe en la

antena no se encuentra entre la lista de permitidas por el nuevo valor del parámetro. Este cambio ocurre en background sin que nos demos cuenta, por lo que detectarlo es realmente complicado.

Los síntomas de que estamos ante esta situación son que al ejecutar el comando de consulta de estado de enlace a la red GPRS:

AT+CGREG?

Obtenemos el resultado:

+CGREG=0,0

Que significa que tenemos los códigos URC desactivados y que no nos estamos registrados en la red y que el terminal ni siquiera está buscando un operador para conectarse, es decir, que el servicio GPRS se encuentra desactivado.

Además, si ejecutamos el comando de comprobación de la calidad de la señal:

AT+CSQ

+CSQ: 99, 99

Que indica que no se detecta señal en la antena porque se encuentra escuchando una frecuencia que no es la válida para el operador que nos daría la conexión.

Además, como se trata de un parámetro (el de la banda radio) que queda fijado de manera inmediata en la memoria no volátil no podemos conseguir restaurarlo utilizando el comando AT&F que resetea algunos parámetros de la configuración con sus valores de fábrica.

Por tanto, la única manera de conseguir solucionar el problema de la conectividad con el operador es localizar el parámetro que ha cambiado y asignarle manualmente el valor que permite que se establezca la selección automática de operador.

Para el caso del cambio de frecuencia a detectar este parámetro se encuentra en la configuración extendida como “Radio/Band” cuyos posibles valores se encuentran en la siguiente referencia [66].

La ejecución del comando:

AT^SCFG=”Radio/Band”,”3”,”15”

Donde 3 y 15 son los valores asignados a las frecuencias GSM 900 MHz + GSM 1800 MHz y GSM 900 MHz + GSM 1800 MHz + GSM 850 MHz + GSM 1900 MHz respectivamente, conseguimos que si el comando AT+COPS está configurado para que la selección sea automática (primer parámetro a 0), se restablezca la conexión con el operador correspondiente quedando el siguiente resultado:

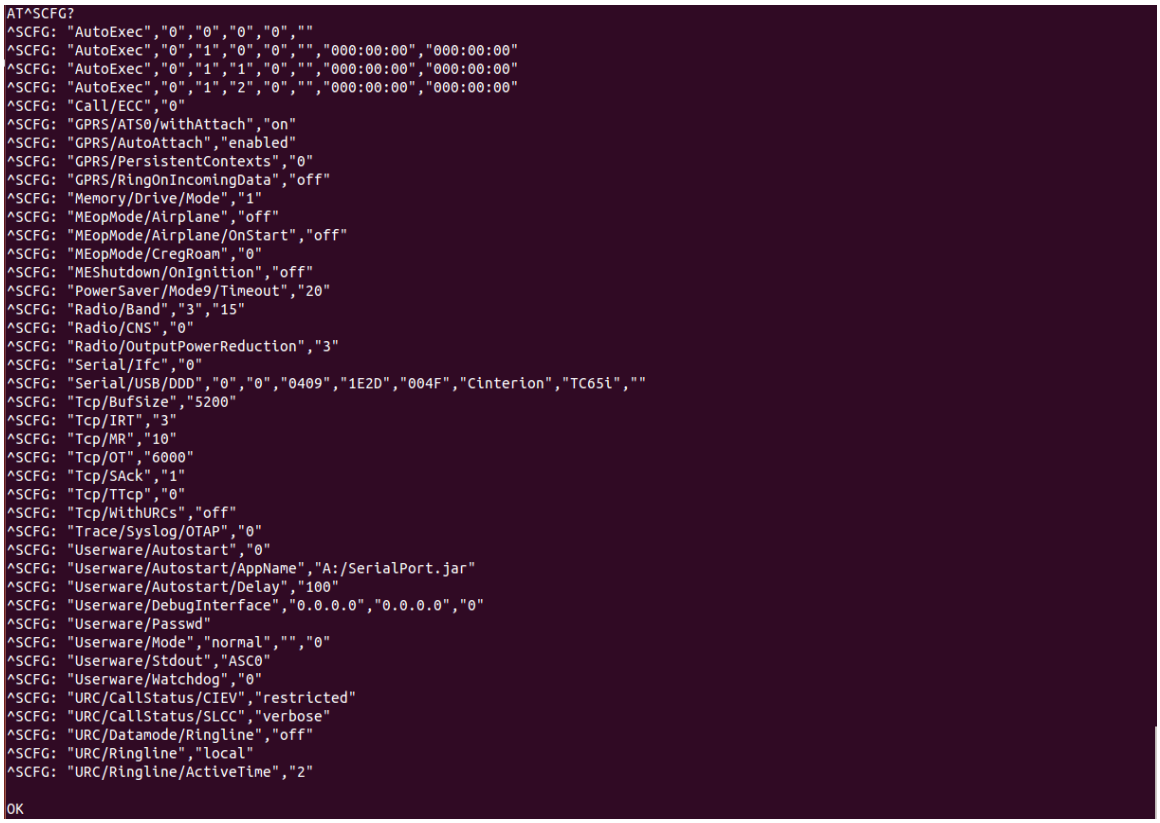

Ilustración 5-22. Captura de pantalla con la ejecución de AT^SCFG? y AT+CGREG?

En general y salvando estos obstáculos la configuración del módem MTX65i vía comandos se puede desarrollar con la ayuda del manual de comandos AT disponible tanto en el SDK como en algunas páginas de internet. El conjunto de comandos AT del dispositivo en cuestión es bastante similar al del modelo anterior y otros del mismo fabricante (MTX65, MTX65i-G, etc.)

6 PROGRAMACIÓN DE APLICACIONES MIDLET

6.1 Introducción al entorno de programación

El entorno de desarrollo integrado utilizado para la codificación de la aplicación es Eclipse.

Al iniciar eclipse lo primero que aparece es la selección del directorio Workspace. Para mejorar la puesta en marcha de proyectos de desarrollo del mismo tipo Eclipse permite que los englobemos dentro de un mismo directorio de trabajo (Workspace). De esta manera cada vez que iniciemos un Workspace no solo encontraremos nuestros proyectos sino que también se tendrá la configuración que utilizamos la última vez, de forma que nos facilite el desarrollo.

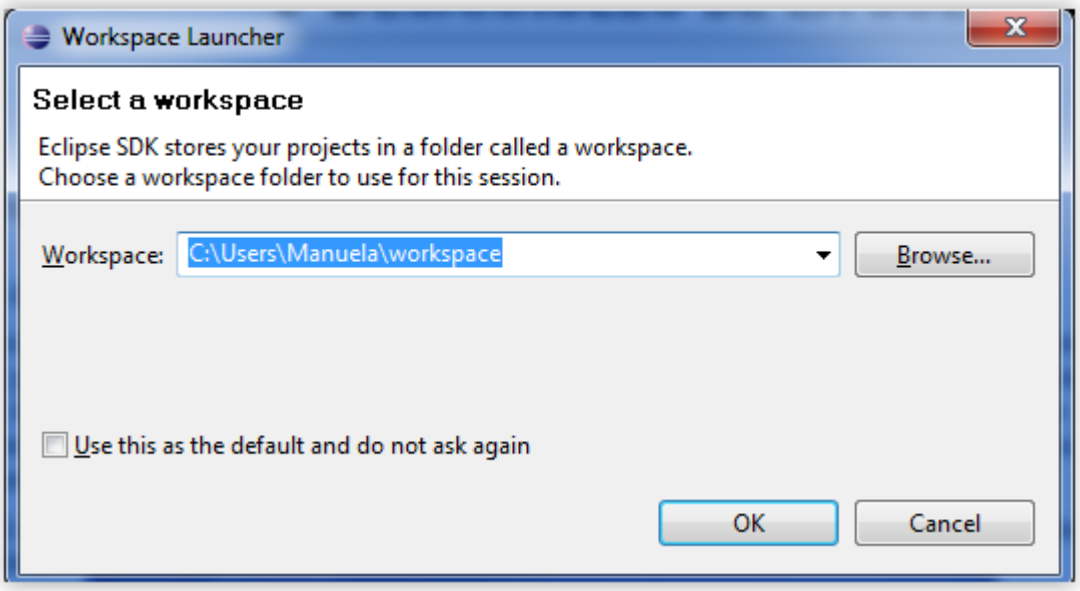

Ilustración 6-1. Selección del Workspace

Lo habitual es escoger el directorio de trabajo que el propio Eclipse propone ya que si elegimos una ruta más extensa podemos tener problemas de compilación y similares.

El entorno de trabajo de Eclipse se divide en varios paneles o *vistas* como por ejemplo "Navigation" o "Package Explorer". Al conjunto de paneles que tengamos en pantalla en un momento determinado se le llama Perspectiva.

Para crear un proyecto cualquiera podemos desde "File" -> "New", en la parte superior, o desde el Package Explorer con el click derecho o Ctrl+N.

Al lado de esta vista tenemos el Área de edición, que es donde escribiremos el código de nuestros proyectos. En la parte derecha tenemos la vista esquemática que muestra un esquema del documento que esté codificando el editor en ese momento.

Por último, en la parte inferior disponemos un panel de información en el que se nos muestran por ejemplo los errores en compilación o los warnings.

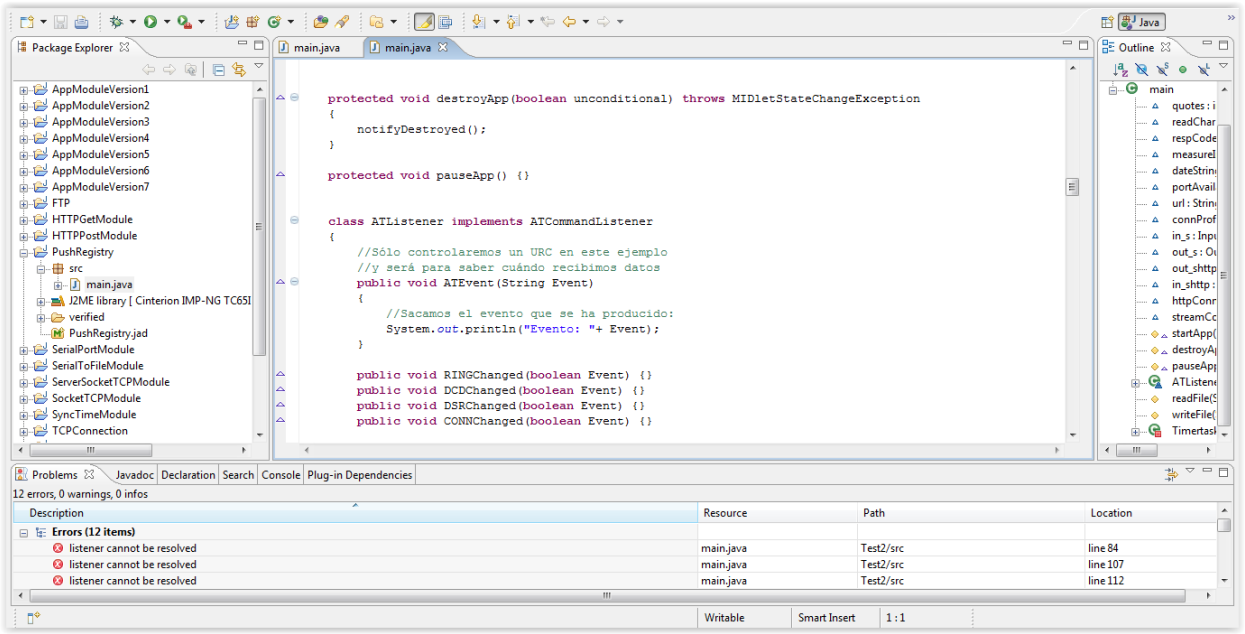

Ilustración 6-2. Vista general Eclipse

Si queremos quitar, cambiar o añadir alguna vista podemos seleccionar Window → New Window y elegirla.

6.2 Introducción a la programación de aplicaciones MIDlet [67]

MIDP (Mobile Information Device Profile) [68] se trata de un perfil específico dentro de Java que permite desarrollar aplicaciones para dispositivos inalámbricos enmarcado en la configuración del CLDC (Connected Limited Device Configuration) que a su vez se encuentra dentro de Java ME. El subconjunto dentro de los posibles MIDP que implementaremos será IMP-NG (Information Mobile Profile – Next Generation) [36].

A las aplicaciones que tienen perfiles MIDP se las denomina aplicaciones MIDlet y están orientadas a tratarse de aplicaciones inalámbricas.

Éstas tienen una estructura de código en las que la clase a implementar (Main por ejemplo) hereda de la clase MIDlet y en ella no existe un método principal sino que cuenta con métodos que es obligatorio incluir para llevar a cabo su ciclo de vida consistente en tres estados:

- Pausa: El MIDlet se ha iniciado y en reposo. No se puede pasar a este estado si se está utilizando alguno de los recursos compartidos.

- Activo: El MIDlet está funcionando normalmente.

- Destruir: El MIDlet ha liberado todos sus recursos y ha terminado. Este estado solo se produce una vez.

Cada estado está respaldado por los siguientes métodos:

- pauseApp (): Aquí el MIDlet debe liberar todos los recursos temporales y esperar.

- startApp (): El MIDlet comienza la ejecución y se inicia la utilización de los recursos (que también se puede hacer desde el constructor de la clase). Tenemos que asegurarnos que se completa correctamente la realización de las acciones de este método antes de pasar al de destrucción.
- destroyApp (): Guarda cualquier estado en el que se encontrase la aplicación y libera todos los recursos. Este método también es llamado si se ejecutan ciertos comandos AT dentro de la aplicación, como por ejemplo AT^SMSO, AT+CFUN, etc. (Consultar Relación de Comandos AT relevantes)

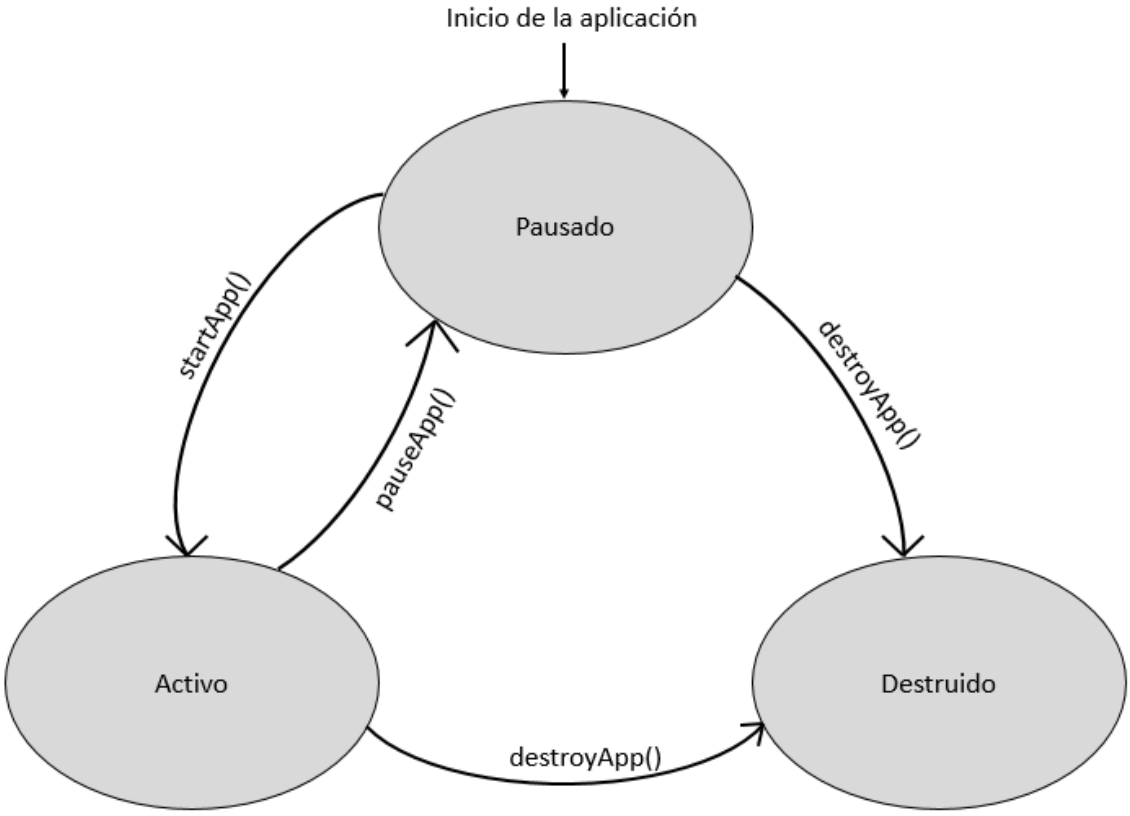

Ilustración 6-3. Estados y métodos de cambio de estado básicos en aplicaciones MIDlet

Además de éstos que es necesario incluir de manera obligatoria, adicionalmente podemos incluir la definición de los siguientes métodos heredados de la clase MIDlet:

- notifyDestroyed (): Notifica al software de gestión de aplicaciones que se va a proceder a la destrucción de la aplicación. Es la única manera de poner fin a un MIDlet y se utiliza dentro del método destroyApp (). No debe ser llamado antes.
- notifyPaused () – El MIDlet notifica al software que da soporte a la gestión de aplicaciones que se ha detenido la aplicación. Se implementa en el método pauseApp ().
- resumeRequest (): El MIDlet notifica al software de gestión de aplicación que se reinicie.

A pesar de que no los implementemos en nuestra aplicación explícitamente podemos hacer uso de ellos.

Las acciones que queremos que nuestra aplicación ejecute tendremos que incluirlas principalmente en el método startApp () o que se llame a los métodos que ejecutan dichas acciones desde este método.

Para crear un proyecto que implemente la clase MIDlet en Eclipse podemos utilizar el wizard que gracias a la instalación del SDK se encuentra en "File" → "Project" → "New Project" y nos aparecerá la siguiente ventana:

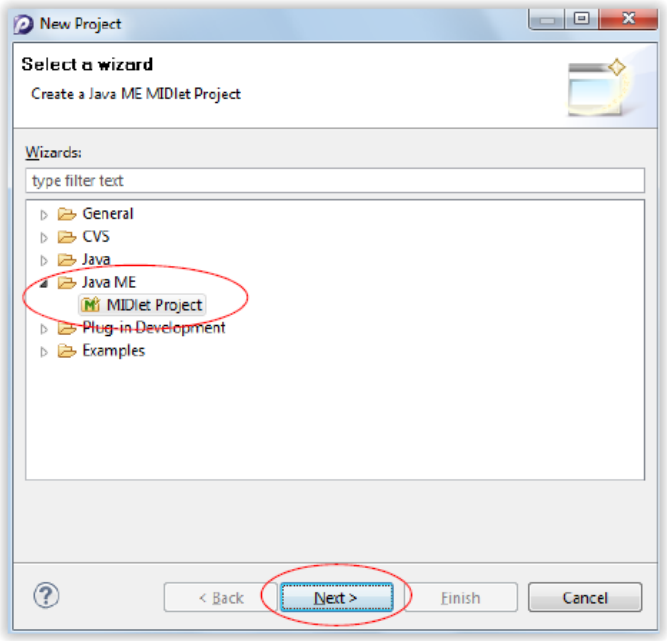

Ilustración 6-4. Captura de selección del wizard para crear un nuevo Proyecto

Después de escoger el Wizard aparecerá la ventana de opciones que se muestra a continuación en la cual debemos dar un nombre a nuestro proyecto MIDlet. Es muy importante asegurarnos de que "IMP_NG_ABC2" se encuentra seleccionado.

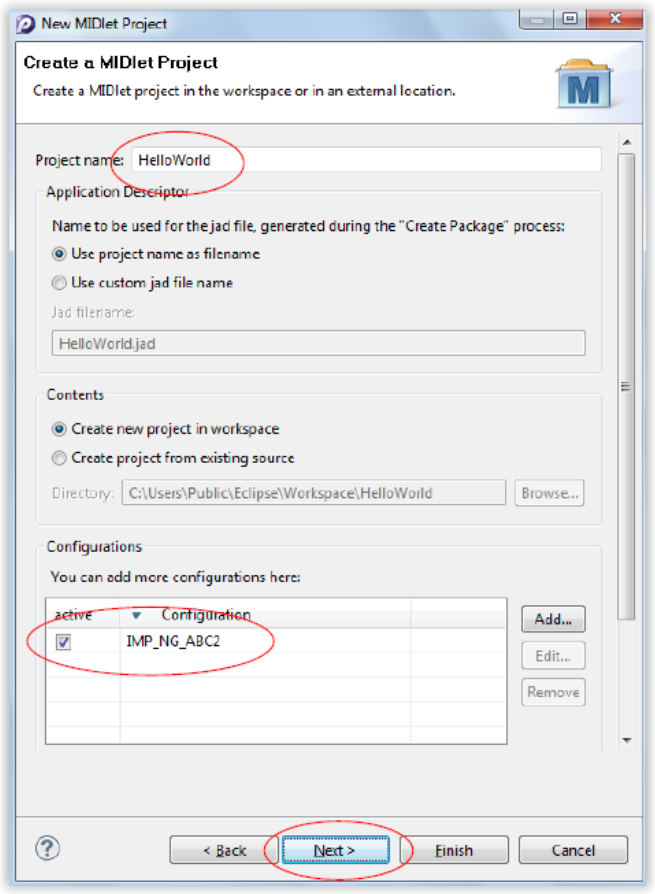

Ilustración 6-5. Ventana de Propiedades del MIDlet creado

Si pulsamos “Next” veremos la siguiente ventana, en la cual es muy importante comprobar que las opciones en “Microedition Configuration” son “Connected Limited Device Configuration (1.1)” y en “Microedition Profile” está seleccionado “Information Module Profile (NG). Esto es crucial ya que si no está la opción correcta seleccionada al ejecutar la aplicación en nuestro dispositivo obtendremos el error “MIDlet Profile Invalid”.

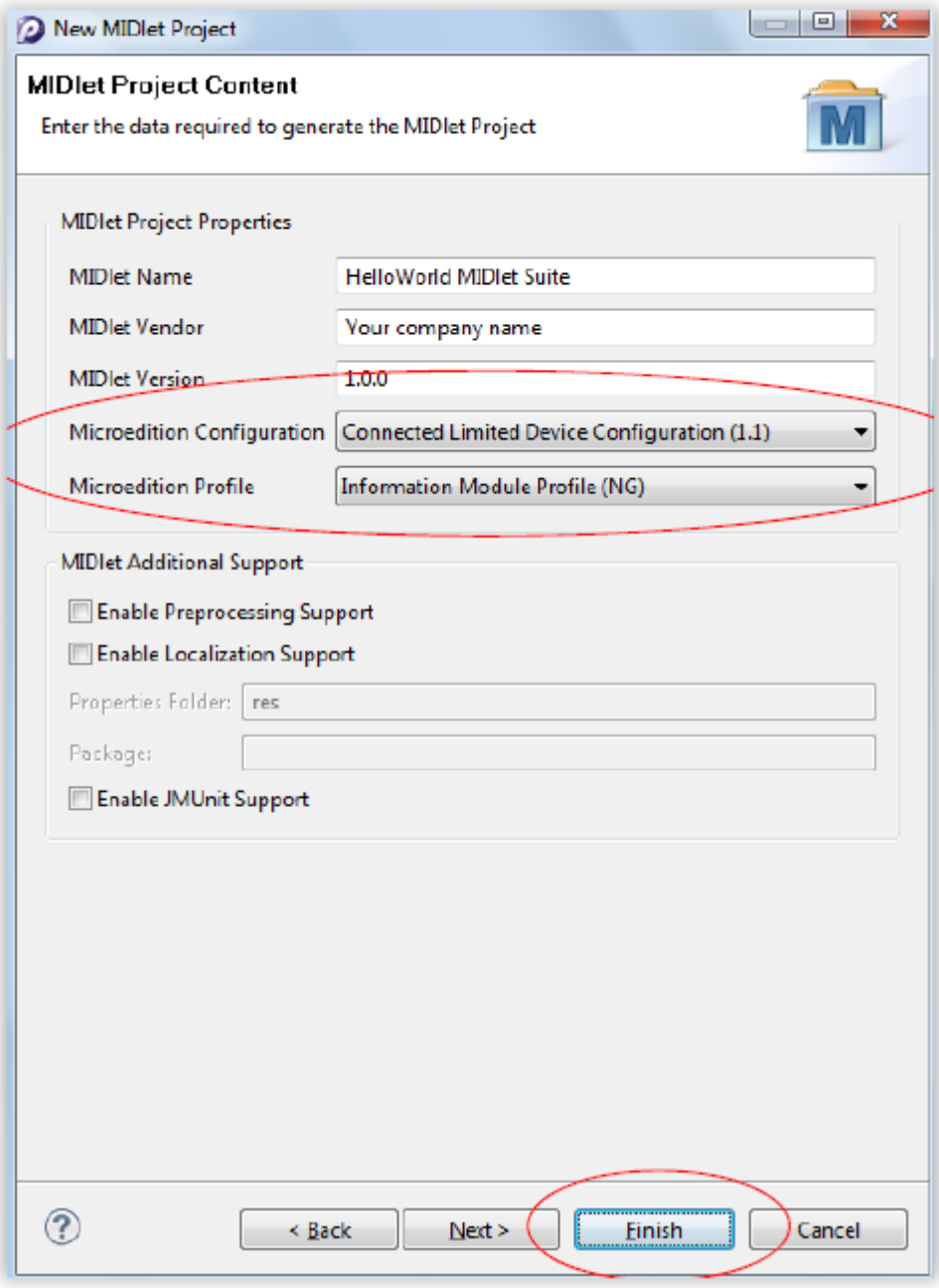

Ilustración 6-6. Captura de la ventana 2 de Propiedades del MIDlet

Clicando “Finish” tendremos un proyecto MIDlet creado en el workspace.

Para finalizar tenemos que cambiar la versión del compilador que se ejecutará en esta aplicación, como se muestra en la siguiente ilustración:

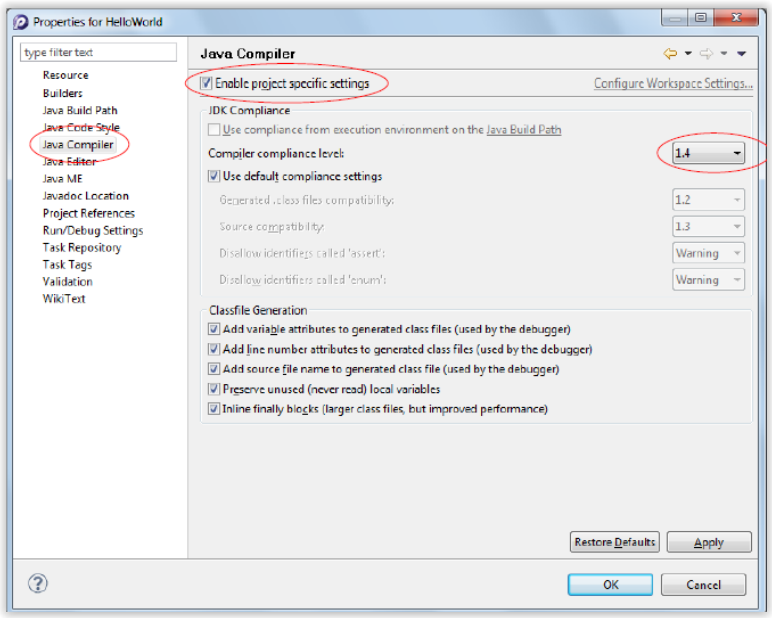

Ilustración 6-7. Propiedades del compilador de Java para cualquier proyecto

Wireless ToolKit incluye un directorio con proyectos MIDlet de ejemplo que podemos importar a nuestro workspace seleccionando “Import” y escogiendo “Existing Projects into Workspace”. En la ventana que aparece tenemos que poner la ruta al directorio del SDK donde se encuentran los ejemplos y seleccionar la opción “Copy projects into workspace”, como podemos ver en la siguiente imagen:

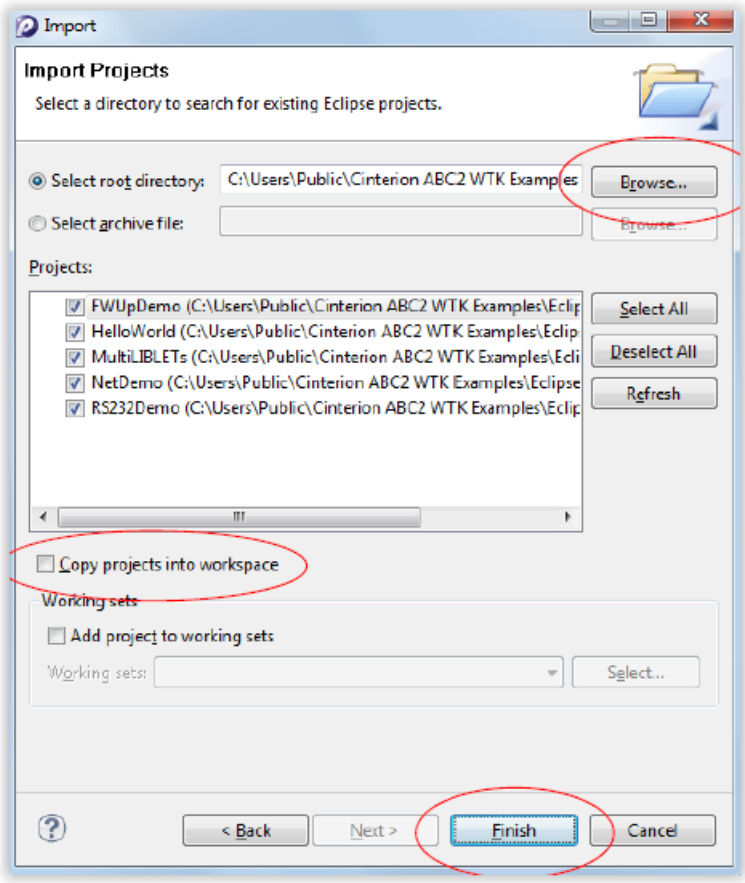

Ilustración 6-8. Captura de la ventana para Importar proyectos existentes

6.2.1 Hello World

El ejemplo más sencillo para mostrar todos los elementos anteriormente descritos en acción es el clásico "Hello World".

Para crear este proyecto desde 0 si no lo hemos importado, podemos utilizar el Wizard disponible como se vio en el apartado anterior (Ilustración 6-4. Captura de selección del wizard para crear un nuevo ProyectoIlustración 6-4 e Ilustración 6-5).

En el fichero HelloWorld.java podemos encontrar la clase main que hereda las características de la clase MIDlet cuyos métodos principales son startApp (), destroyApp () y pauseApp () y deben estar definidos aunque su cuerpo se encuentre vacío, ya que de lo contrario se generará un error ya que estamos heredando de una clase sin implementar sus métodos obligatorios.

Para implementar una aplicación MIDlet que imprima en la salida estándar un mensaje "Hello World!" basta con definir en la clase principal la cadena a imprimir y los métodos citados anteriormente dejando el constructor de la clase vacío. En el método startApp () (que se ejecutará automáticamente sin necesidad de llamarlo nada más arrancar la aplicación en caso de que no llamemos a cualquiera de los otros dos métodos) incluimos la línea de impresión por la salida estándar.

De no incluir nada más la aplicación cumplirá el objetivo e imprimirá "Hello World!", pero si lo ejecutamos veremos que hasta que no reiniciamos el módem no salimos de la aplicación al modo comandos. Esto se debe a que al finalizar la impresión del mensaje no hemos incluido la llamada al método destroyApp () lo que implica que la aplicación no podrá cambiar de estado y se quedará permanentemente sin destruirse. Para solventar esta situación tenemos que añadir al código la línea de llamada al método destroyApp () como se ve en el Anexo.

La codificación ya ha concluido pero es importante antes de compilar comprobar que las propiedades de los perfiles MIDlet están bien ajustadas (Captura de la ventana 2 de Propiedades del MIDlet) y la versión del compilador es correcta (Propiedades del compilador de Java para cualquier proyecto).

Para obtener los archivos necesarios (*.jar* con la aplicación y *.jad* con la descripción del perfil que define la aplicación) compilamos el proyecto completo mediante "Project" → "Build project" y creamos el paquete haciendo click derecho sobre el proyecto elegimos "JME" → "Create package". Los ficheros *.jar* y *.jad* necesarios aparecerán en el directorio "*deployed*" en el interior del proyecto.

Ahora incluimos estos dos ficheros en la memoria interna del módem accediendo como se indica en el apartado Vía Micro USB/USB para acceder a la memoria. Para ejecutarlo debemos iniciar una comunicación serie con el módem (Vía Micro USB/USB para establecer una comunicación serie o Vía puerto serie con un conector DB9 para establecer una comunicación serie) y una vez establecida insertamos el comando para ejecución de aplicaciones indicando la ruta en la que se encuentra el fichero *.jar* de la aplicación.:

AT^SJRA="A:/HelloWorld.jar"

Si tenemos la salida estándar configurada adecuadamente (ASC0 si nos conectamos por puerto serie o USB si nos utilizamos ese conector) aparecerá por pantalla el mensaje "Hello World!".

6.2.2 Thread

En este caso y con los mismos pasos y características que en la creación de la aplicación Hello World, vamos a crear una que sirva de ejemplo de funcionamiento de los hilos.

Para ello creamos igualmente mediante el Wizard un proyecto MIDlet denominado "Thread" que lo que hará será crear una clase "ThreadExample" que hereda de la clase MIDlet y tendrá en su interior definida otra clase que implementará la clase Thread. Thread permitirá un hilo de ejecución independiente dentro de un programa gracias a que implementa la interfaz Runnable (para ejecutar código mientras otra parte del programa sigue ejecutándose). Tenemos dos formas de implementar un hilo. La primera consiste en declarar nuestra propia clase "MyThread" que herede de la clase Thread pero sobreescribiendo el método "*run*" que será el que contenga el código que

queremos que se ejecute. La segunda es crear una clase cualquiera que implemente la interfaz Runnable que es la que aporta la capacidad de ejecución simultánea.

Nosotros en este ejemplo vamos a crear una clase que herede de la clase Thread como se ve en el código del anexo.

Así nuestra clase main del proyecto contendrá las variables necesarias para su ejecución (un tipo boolean para indicar si seguimos ejecutando el hilo o lo paramos y un objeto de la clase ThreadExample), el constructor de la clase y la clase implementada ThreadExample. Ésta contendrá igualmente tanto las variables que se necesiten como el constructor de la clase, además del método *run* que será el código que conformará la acción de la aplicación. En nuestro caso se tratará de un contador que al llegar a cierta cantidad (bucle for) interrumpirá el hilo poniendo la variable runThread a false como se puede ver en el anexo.

La ejecución de esta aplicación tras ser compilada, empaquetada y puesta en la memoria del módem será:

6.2.3 Timer

La clase Timer en realidad se trata de un hilo ejecutado en segundo plano que pasado cierto tiempo indica que ha expirado el timeout establecido. Se utiliza por ejemplo para ejecutar tareas en un instante de tiempo determinado o para repetir tareas cada cierto tiempo.

En este ejemplo se implementa un contador que imprime cada segundo los segundos que lleva contando el timer. Para ello en primer lugar es necesario implementar una clase que herede de la clase TimerTask (que es la que se encarga de definir un método run propio que se ejecutará cuando el temporizador expire). En esta clase llamada Monitor en nuestro caso heredamos de la clase TimerTask y en el método run incluimos una impresión del valor actual de un contador que previamente se ha incrementado en 1 unidad.

Ahora en el método startApp () de la clase principal "main" creamos un objeto de tipo Monitor y otro de tipo Timer. Con la siguiente línea iniciamos el TimerTask de manera que en el intervalo de tiempo indicado como tercer parámetro se ejecutará el método run () del objeto "mon":

```
timer.schedule(mon,0,1000);
```

Así se imprimirá por pantalla el valor de un contador incrementado cada segundo.

7 DESARROLLO DE LA SOLUCIÓN

Tras lo aprendido con los breves ejemplos anteriores nos proponemos comenzar a implementar la aplicación final objeto de este proyecto. El enfoque del desarrollo ha sido la construcción de pequeños módulos de código independientes que realizarán las tareas más simples en que se puede dividir la aplicación (como por ejemplo el envío de mensajes HTTP) de manera que al ser ejecutadas con ciertos parámetros de entrada proporcionasen la salida requerida. Con este modelo de caja negra a la hora de unir las partes diseñadas nos aseguramos una acotación de los errores a la hora de su depuración total así como una ejecución más comprensible y secuencial.

La división de estos módulos se ha realizado en primer lugar en base al segmento de comunicación en el que se encuentre la aplicación (comunicación serie entre módem y contador, comunicación entre el módem y el servidor central) o si se encuentra en tareas de procesado interno (ejecución del Timertask)

7.1 Comunicación Módem – Red

7.1.1 Time Sync Module

La aplicación en primer lugar debe sincronizar su hora actualizándola con el valor que se le proporcione cuando se una a la red. Esto se consigue con la posibilidad de ejecutar comandos AT desde dentro del código en Java de manera que con un comando denominado AT^SIND consigamos los datos de la hora actual en el meridiano de Greenwich (GMT) junto con el desfase del huso horario en el que se encuentre el módem medido en cuartos de hora, así como un indicador de si se encuentra activo el horario de verano en la localización actual.

De esta manera obtenemos la hora pero es necesario fijarla como la hora del módem gracias al comando AT+CCLK que se utiliza para este fin.

El único problema que se encuentra en esta fase es que la información devuelta por el primer comando no es compatible para ser directamente introducida como cadena al segundo comando, por lo que la información horaria debe ser procesada para poder introducir el formato adecuado en el comando AT+CCLK. Esto se consigue con la ejecución de un bucle que recorre la cadena recibida de la red en busca de la fecha y hora que se va a introducir, guardándola en otra cadena que será el parámetro de entrada al segundo comando.

7.2 Comunicación Módem – Contador

7.2.1 Serial Port Module

Ahora nos encontramos en la fase de interacción que se dará entre el módem y el contador vía comunicación serie.

En este caso lo primero que debemos comprobar es qué nombre podemos asignar al puerto serie correspondiente al contador. Se trata de un dato fundamental para establecer la comunicación entre el módem y el contador ya que todos los flujos abiertos (de entrada, salida de datos) deben conocer el nombre del puerto al que se encuentran ligados.

Para ello basta ejecutar el método:

```
String ports = System.getProperty("microedition.commports");
```

Que nos devolverá una lista de cadenas separadas por comas que contiene un listado predefinido por el módem de los nombres disponibles para asignar a los puertos conectados (COM0 y COM1 dado que el módem solo tiene dos interfaces puerto serie ASC0 y ASC1).

Escogiendo uno de los valores devueltos tendremos ya listo un nombre que asignar al puerto serie que se encuentre conectado al contador.

El siguiente paso es abrir los flujos de entrada y salida de información que son punteros a las zonas de memoria donde vamos a insertar los datos al escribir en el flujo o de donde vamos a leer en el caso contrario.

Todos los tipos de flujos que se pueden utilizar en Java ME (StreamConnection, HttpConnection, ServerSocketConnection, etc.) necesitan ser instanciados con el mismo procedimiento que detallaremos a continuación.

Lo primero es crear un objeto de la clase del flujo que nos dispongamos a abrir en nuestro caso StreamConnection (debido a problemas con la utilización de CommConnection que se detallarán posteriormente).

```
StreamConnection streamConn = null;
streamConn = (StreamConnection)Connector.open("comm:"+
                    portAvailable+";baudrate=115200;blocking=off");
```

Con la segunda línea lo que hacemos es utilizar la clase Connector que crea objetos de tipo Connection y cuyo método *open* abre la posibilidad de comunicación entre los dispositivos. En este método el único parámetro de entrada serán los valores de configuración necesarios para la comunicación establecida (baud rate, nombre del puerto, etc.)

Pero para que haya comunicación es necesario que se habran flujos tanto de lectura como de escritura que se encuentren vinculados al flujo de comunicación principal. Esto se consigue haciendo:

```
InputStream in_stream_serial = streamConn.openInputStream();
OutputStream out_stream_serial = streamConn.openOutputStream();
```

De manera que si ahora queremos leer o escribir solo tenemos que llamar a los objetos correspondientes de tipo InputStream u OutputStream con los métodos disponibles en cada uno a tal efecto, como se puede ver en la siguiente línea:

```
int currentCharacter = in_stream_serial.read();
bufferOut = ":01MRSendMeMeasurementData/n";
out_stream_serial.write(bufferOut.getBytes());
```

Y con una secuencia organizada de lectura y escritura del flujo de conexión quedaría definida la comunicación serie.

7.3 Comunicación Módem – Servidor Central

La comunicación con la central la podemos realizar de tres maneras diferentes: vía socket TCP si queremos que la comunicación sea a nivel de la capa de transporte o HTTP vía GET/POST si queremos llegar hasta la capa de aplicación.

7.3.1 Socket TCP Module

Uno de los objetivos consiste en que sea posible requerir desde el servidor una medida asíncrona en cualquier momento. Esto solo es posible si, a la vez que el módem ejecuta el hilo principal del programa con el TimerTask funcionando, en otro hilo independiente tenemos un servidor de cualquier tipo esperando permanentemente esa conexión entrante. Como ya se ha comentado el número de clases incluidas en la librería JME es reducido dado

que se trata de una versión de Java destinada a dispositivos móviles con pocos recursos. Es por este motivo que la única clase servidor que se adapta a nuestras necesidades es la denominada ServerSocketConnection cuyo cometido es iniciar un socket de escucha que detecte un posible inicio de sesión desde el servidor.

Análogamente al caso de la comunicación serie es necesario crear un objeto de tipo Stream que abra un socket de escucha en el puerto elegido. Para ello incluimos en el método startApp () las siguientes líneas tras la sincronización de fecha y hora con la red:

```
serverSocketConn = (ServerSocketConnection)
                    Connector.open("socket://:23"+connProfile);
socketConn = (SocketConnection) serverSocketConn.acceptAndOpen();
```

Con esto abrimos el flujo de la posible conexión en el puerto indicado y bloqueamos el proceso hasta que se establezca alguna conexión con el socket de escucha. Es muy importante que en la ejecución del programa final incluyamos un hilo independiente del principal con esta característica porque en caso contrario la aplicación no saldría de este punto y no ejecutaría la petición de medidas periódicas.

Además de indicar que se trata de una conexión de tipo socket y el puerto el método open necesita que incluyamos en la cadena la información del APN que vamos a utilizar para la conexión con el exterior vía GPRS. Por tanto en la cadena connProfile incluiremos los siguientes valores de configuración:

```
private String connProfile = ";bearer_type=gprs;access_point=wlgdsp.com;" +
            "username=well;password=well;timeout=-1;";
```

Tal y como se comentó en el caso serie es necesario abrir también los correspondientes flujos de entrada y salida para lectura y escritura.

7.3.2 HTTP vía GET Module

Los mensajes HTTP también utilizan una clase de tipo Connection para establecer un flujo que soporte el intercambio de mensajes de petición y respuesta entre el módem y el servidor:

```
HttpConnection httpConn = (HttpConnection)Connector.open(url + connProfile);
```

Que abre un flujo de tipo Connection en el que utilizamos como en los casos anteriores el método open y cuyo parámetro incluye igualemente el perfil de la conexión GPRS con las propiedades del APN anteriormente descritas.

Como se explicó en el apartado sobre HTTP podemos utilizar dos métodos en este protocolo según donde viaje la información (cabecera en el caso GET por lo que quedaría visible en la URL o en el cuerpo si usamos POST).

Esta elección con la siguiente línea:

```
httpConn.setRequestMethod(HttpConnection.GET);
```

A partir de aquí como ocurre en todas las conexiones tendremos que añadir el código que habilita el uso de streams tanto de entrada como de salida para la lectura y la escritura de datos con las características del protocolo que utilicemos.

Una vez hayamos escrito con el método write en el flujo de salida la línea que envía el mensaje de petición es:

```
int respCode = httpConn.getResponseCode();
```

Este método nos devuelve el código que nos envíe en respuesta el servidor HTTP (200 OK si todo va bien y otro en caso contrario).

7.3.3 HTTP vía POST Module

En este caso el código es idéntico al caso HTTP GET salvo al fijar el método que tenemos que indicar que en este caso es POST:

```
httpConn.setRequestMethod(HttpConnection.POST);
```

Además aquí podemos fijar algún parámetro adicional como el Content-type:

```
httpConn.setRequestProperty("Content-Type",
                    "application/x-www-form-urlencoded");
```

En nuestra aplicación tal y como explicaremos más adelante, utilizaremos una combinación del módulo HTTP vía POST en el caso del envío de información desde el módem al servidor ya sea de manera síncrona o asíncrona porque así evitamos que los datos viajen en la cabecera apareciendo en la URL.

En caso de que el que inicie la conexión sea el servidor para solicitar una medida instántanea utilizaremos el socket TCP mencionado anteriormente de manera que no haya que esperar a que expire el temporizador para obtener los datos.

7.4 TimerTask Module

Este módulo es el alma de la aplicación ya que es el encargado de que se ejecute periódicamente de manera sencilla. Tal y como hicimos en el ejemplo Timer creamos una clase que herede de TimerTask cuyo método run va a estar compuesto por las llamadas a los módulos de lectura del contador vía puerto serie y envío de medidas por HTTP.

```
TimertaskClass timertaskObj = new TimertaskClass();
timer = new Timer();
timer.scheduleAtFixedRate(timertaskObj, 0, measureIntervalTimeout);
```

Desde el momento en el que se ejecuta el método scheduleAtFixedRate el timerTask está en funcionamiento dado que el segundo parámetro que es el retardo en el inicio del temporizador está a 0.

Además si el servidor devuelve algún código de respuesta de error o el eco recibido de los datos enviados es incorrecto, guardaremos en un fichero los datos enviados para su posterior recuperación, utilizando un método de escritura en un stream de fichero que se definirá a continuación.

7.5 File Backup Module

Este es un módulo adicional en el módem se encarga de guardar los datos de las medidas en un fichero de respaldo en caso de no recibirse correctamente en el servidor.

El módulo utiliza la apertura de un flujo en un fichero que de no existir se crea y en caso contrario se escribe al final del mismo.

Las líneas principales de este módulo son:

```
FileConnection fileConn = (FileConnection)
                    Connector.open("file:///a:/backup.txt");
OutputStream out_stream_file = fileConn.openOutputStream(sizeFile);

out_stream_file.write(data.getBytes());
```

En la apertura del flujo de escritura el parámetro de entrada del método es el tamaño del fichero. Con esto conseguimos que el descriptor de fichero se localice al final del contenido del mismo evitando la sobreescritura.

7.6 Application Module

La aplicación final se constituye de cada uno de estos módulos definidos como métodos y además incluye código adicional para la interacción entre los mismos.

En el siguiente diagrama de flujo podemos ver la ejecución que sigue el programa de manera general:

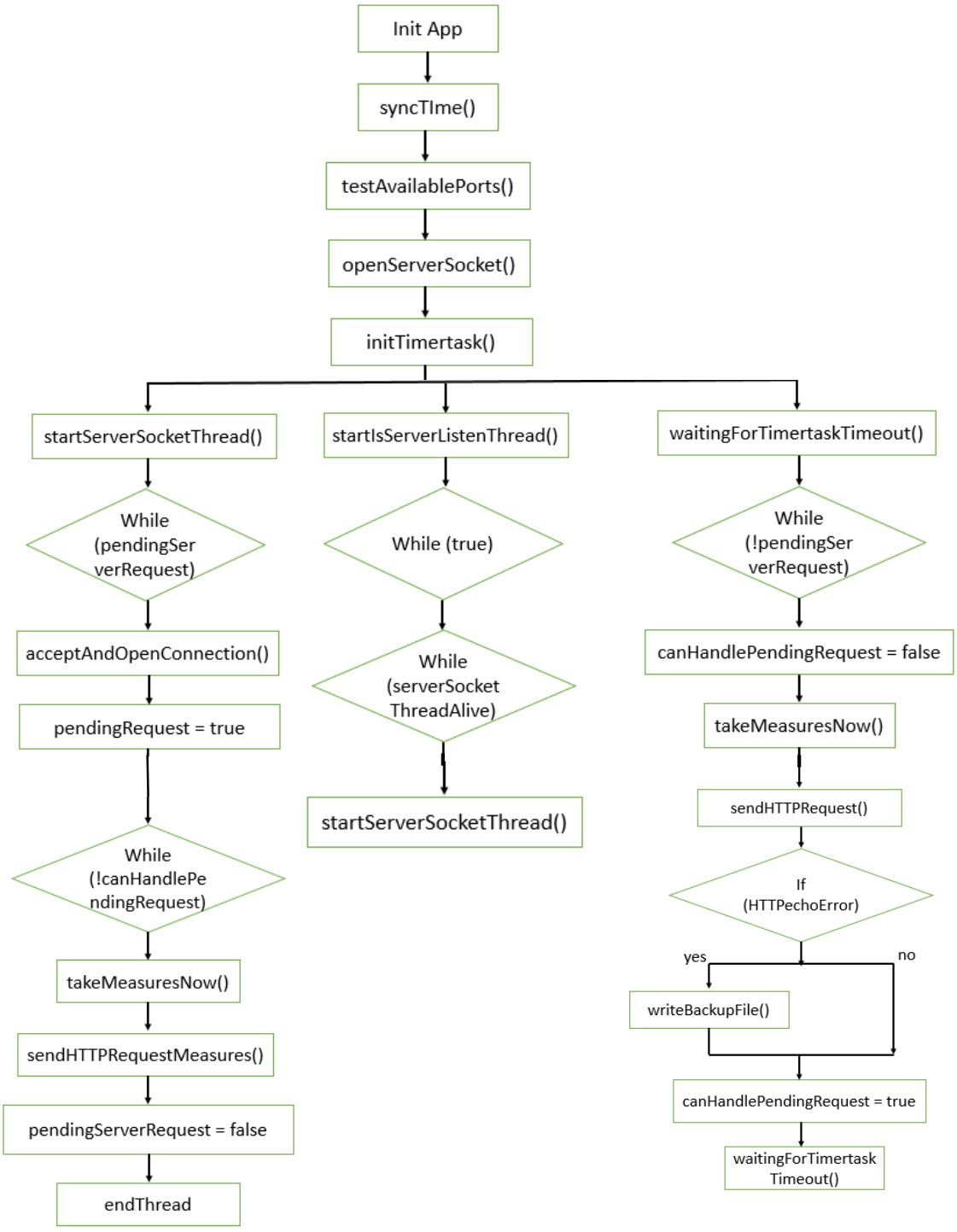

Ilustración 7-1. Diagrama de flujo de la aplicación final

La secuencia de la aplicación completa comienza con la sincronización de la fecha y hora a través de la red para posteriormente comprobar los puertos disponibles para solicitar medidas del contador. Acto seguido se inicia el socket de escucha en el cual un servidor capta las conexiones entrantes que pueda producirse asíncronamente desde la central para solicitar medidas en cualquier momento. Posteriormente iniciamos el TimerTask cuya actuación consiste en solicitar del contador las medidas y enviarlas en un mensaje HTTP Request que de no recibir la respuesta adecuada propicia que se guarden las medidas en un fichero de backup que posteriormente se recuperaría de la memoria del dispositivo.

Una vez terminadas las acciones del timertask definido la aplicación permanece en espera hasta que se produce una petición de datos por parte del servidor. Si ésta no llega el temporizador del timertask expira y las tareas de solicitud de medidas y envío de datos se repiten periódicamente.

Un aspecto fundamental en la aplicación es la utilización de hilos para mantener el socket de escucha permanentemente a la espera. Si se recibe una petición el hilo termina su ejecución y termina lo que en principio haría que no se pudieran recibir más peticiones de medidas hasta que no terminase el temporizador del timertask. Este no es el objetivo definido como parte de la aplicación, por lo que la solución implementada incluye una nueva clase con las cualidades de la clase Thread cuya misión es comprobar permanentemente que el hilo que contiene socket de escucha se encuentra activo. De no ser así, lo reinicia de nuevo de manera que siga pendiente.

Otra de las claves en el diseño de la aplicación es que existen varios métodos a los cuales hilos independientes (como el de mantener el socket de escucha activo y el principal que contiene el Timertask) pueden llamar simultáneamente lo que produciría indeterminaciones en la apertura y cierre de flujos de entrada y salida. Para dar una solución robusta al problema de los métodos compartidos por los distintos hilos se han creado dos variables semáforo cuyo valor se comprueba siempre que se va a llamar a alguno de los métodos en cuestión.

Estas variables booleanas se han denominado pendingServerRequest que se activa en cuanto se recibe una petición asíncrona por parte del servidor para obtener una medida instantáneamente y canHandleServerRequest que se activa cuando el hilo principal llega a una comprobación de la primera variable y la ve activa. De esta manera se avisa de un hilo a otro si es consciente de que necesita llamar a los métodos implementados.

La visión general del contenido del fichero main.java cuya compilación genera el AppModule.jar que se ejecutará en el módem es la siguiente:

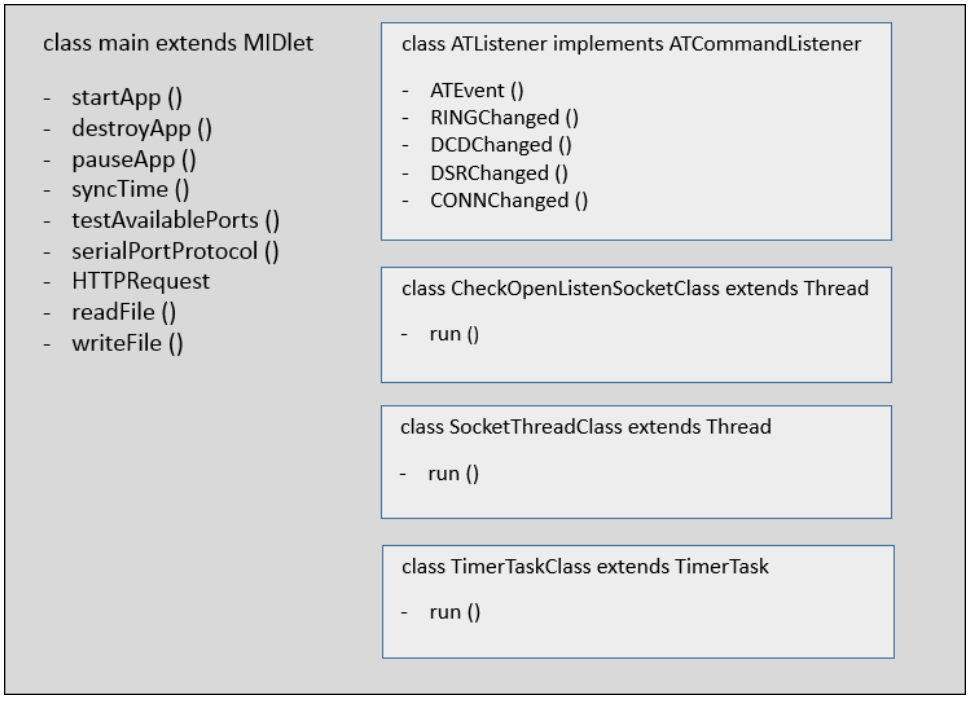

Ilustración 7-2. Diagrama de las clases y métodos incluidas en el fichero principal de la aplicación

8 PRUEBAS

Para probar la lectura de los datos del contador desde el módem dado que finalmente no se ha contado con él, la parte en la que el módem se comunica por puerto serie para recibir medidas en respuesta ha sido emulada con la comunicación serie establecida entre el módem y Minicom. A través del mismo se han insertado los datos que aparecerían en el flujo de entrada de la comunicación serie con el módem si fuera el contador el que se encontrase al otro lado, de manera que comprobamos que las tramas enviadas y recibidas son correctas y que la dinámica de flujos funciona de la manera deseada.

En el caso del envío de datos al exterior es necesario habilitar un servidor que responda a los mensajes HTTP Request que el módem envía con los datos. Con la meta de comprobar también si no solo el servidor recibe el mensaje sino que recibe los datos tal y como se han enviado (errores en las medidas podrían conllevar pérdidas de facturación por ejemplo), el servidor debe enviar como mensaje de respuesta el eco del recibido. En caso de comprobarse que no se ha realizado de la manera adecuada el módem guardará estos datos en un fichero que se almacenará en la memoria del dispositivo para su posterior recuperación.

Por último, para iniciar una conexión externa desde el servidor hacia el módem y conseguir una medida asíncrona del contador es necesario probar que el servidor habilitado en el módem escucha y responde a la conexión entrante del servidor (y solo a la de éste ya que sería una vulnerabilidad que cualquier otro agente solicitará una medida del contador y se le proporcionase). Debido a la configuración de la red de la empresa que ha proporcionado la SIM con la tarifa de datos no ha resultado trivial la prueba de este aspecto, como se detallará en el siguiente apartado.

8.1 Herramientas de pruebas

Para las pruebas descritas con anterioridad, además de las herramientas utilizadas a lo largo del proyecto como el Minicom se han habilitado algunas adicionales que han permitido realizar una batería de pruebas al funcionamiento de la aplicación en general.

Para probar el envío y recepción de mensajes HTTP se ha habilitado un servidor con una instancia virtual de Ubuntu 14.04 en la que se ha instalado una versión de Apache con el intérprete de PHP para este fin. El servidor se encuentra a la espera de recibir mensajes POST y con la intención de comprobar que el campo de datos recibido es correcto se ha incluido un código en PHP que simplemente envía un eco del cuerpo del mensaje recibido, de tal manera que si en la trama de respuesta el módem detecta que las cadenas de datos son distintas, se inicia el proceso de backup en el fichero de datos destinado a tal efecto.

En el caso de las pruebas de la conexión entrante los inconvenientes se han debido en su mayoría a la utilización de un APN de la intranet de la empresa en cuestión. La topología de la red se compone de dos VPN, la perteneciente a una de las oficinas de la empresa y la del proveedor de servicio que interconecta las distintas VPN a otra VPN. En las pruebas de conectividad realizadas con el servidor HTTP esta configuración no suponía ningún problema dado que era el módem desde el interior de la red el que realizaba una petición al exterior y recibía un mensaje en respuesta. Pero al tratarse ahora del caso contrario en el que habilitamos el módem como un servidor que espera recibir peticiones nos encontramos con la seguridad necesaria en este tipo de redes. Las conexiones entrantes que no tienen una petición realizada desde dentro con anterioridad son rechazadas sin posibilidad de llegar hasta el socket de escucha del módem.

Llegar a esta conclusión fue posible debido a que al imprimir la dirección en la que estaba escuchando el socket, se imprimía una dirección privada a la que es imposible realizar peticiones desde el exterior.

Una vez descubierto este inconveniente mientras se probaba el módulo del servidor socket se informó a la empresa con la intención de conseguir acceso a la intranet para que, estando ambos servidor y cliente en la misma red se probase dicho aspecto.

Finalmente la solución pasó por utilizar un acceso especial a través de SSH a la red que el proveedor de servicio proporciona a Wellness, quedando de esta manera dentro de la red virtualmente lo que implica que se permitía el inicio del socket sin problema utilizando la IP privada.

8.2 Pruebas realizadas a los componentes

Se han realizado pruebas a todos los niveles de los diferentes componentes siguiendo el modelo de caja negra en el cual solo comprobamos las salidas y entradas del componente de manera que se cumplan los requisitos que han de cumplir en su futura integración con el resto de módulos.

En primer lugar pruebas unitarias de los componentes gracias a la inclusión de pequeños fragmentos de código adicionales hemos comprobado que cada componente cumple la función individual para la cual ha sido diseñado.

Del mismo modo se han realizado pruebas de integración. A medida que se van enlazando los módulos de forma progresiva vamos comprobando que se mantiene la integridad del sistema y que los resultados son los correctos.

Por último las pruebas de sistema en las que tras la integración de todos los módulos y la adición de código necesario que enlace los componentes comprobamos que la aplicación cumple el objetivo al completo.

Tras esta batería de pruebas hemos comprobado que el sistema cumple con los requisitos que se especificaron en un primer momento con lo que damos por finalizada la fase de desarrollo.

9 RESULTADOS

Los resultados obtenidos en líneas generales son satisfactorios dado que se han alcanzado los objetivos marcados de comunicación y transferencia de información entre el dispositivo de medida y el servidor mediante el módem MTX65i. Además se ha conseguido con una aplicación ligera que cumple los requisitos.

Tras el conjunto de pruebas realizadas que se pueden calificar como exitosas ya que nos muestran que la aplicación cumple los hitos marcados inicialmente por la empresa que propuso el proyecto (salvo los relacionados con la autenticación con el contador y las pruebas físicas con éste que por motivos ajenos a este proyecto no se han podido realizar debido a la falta del dispositivo en cuestión).

No solo se cumple el propósito general si no que a medida que se ha desarrollado la aplicación se ha seguido un modelo de componentes para la programación de los distintos módulos que ha permitido realizar pruebas unitarias a los mismos. De esta manera tras asegurarnos que de manera individual los componentes cumplen con lo que se les exige es cuestión de tiempo que en la integración de los componentes en un solo flujo consigamos resultados idénticos.

A partir de ahora se pueden realizar pequeñas mejoras que no solo hagan nuestra aplicación más eficiente sino que además permitan que sea compatible con el mayor número de dispositivos del mismo tipo como contadores de otros suministros por ejemplo agua o gas.

10 CONCLUSIONES Y PROPUESTAS DE MEJORA

10.1 Conclusiones

Tras el desarrollo del software que permite la recolección de datos gracias a un dispositivo asequible y una aplicación diseñada para sistemas móviles hemos obtenido una aplicación que cubre las necesidades de comunicación de datos propuestas y añade algunas funcionalidades útiles como la escritura del fichero de backup o el eco por parte del servidor HTTP de manera que podamos comprobar si se han recibido bien los datos.

Mediante el desarrollo de esta aplicación se han adquirido conocimientos en todos los aspectos que el proyecto ha requerido.

En primer lugar el manejo y puesta en marcha de un dispositivo físico como es el módem MTX65i.

Trabajar con un dispositivo como éste supone un reto adicional ya que implica tener un amplio conocimiento del mismo para poder hacer un uso adecuado de sus funcionalidades y responder ante los problemas de configuración y operación que se producen en el mismo. Para ello es necesario dedicar un gran número de horas a los extensos manuales y extraer una visión general que nos ayude durante el transcurso del proyecto así como a darnos fluidez en la posible búsqueda y resolución de problemas que nos podamos encontrar.

Otro de los aspectos fundamentales desarrollado a lo largo este proyecto ha sido la parte de programación del software. Al tratarse de una versión de Java y dados los escasos conocimientos en principio de este lenguaje de programación supone un reto adicional adecuarse a una versión específica del mismo adaptada a dispositivos móviles, y más aun sumando el hecho de las clases definidas específicamente para el módem en cuestión disponibles gracias a la librería del SDK.

Además hay que tener cuenta que la solución a implementar se encontraba en un nivel que requería nociones avanzadas de la dinámica de los protocolos de comunicación utilizados y distintos estándares (TCP, HTTP, RS232) para poder implementarlos de la manera adecuada en el código de nuestra aplicación interactuando entre sí en el momento oportuno.

Por supuesto gracias a estos aspectos llegamos a la ingeniería del software en cuanto a diseño de la solución. Es uno de los aspectos más importantes ya que precisamente en el diseño de la aplicación, su modularidad y secuencialidad radica el éxito de su funcionamiento. Por ejemplo al trabajo realizado en el diseño debemos la idea de los distintos hilos de ejecución en el programa que permiten realizar acciones simultáneas a pesar de que una de las características de las aplicaciones móviles es precisamente la simpleza de las mismas.

Finalmente y ya en la fase de integración y puesta en funcionamiento de la topología al completo ha sido un aspecto fundamental la perseverancia en la búsqueda de soluciones lo que nos ha llevado a la consecución de una versión funcional del software solicitado.

Aun así es posible realizar ciertas mejoras en el mismo lo que, lejos de ser un punto desfavorable de la misma, ofrece una línea de trabajo a seguir para convertir esta aplicación en una buena opción para el despliegue de dispositivos ligeros con aplicaciones escalables y multiplataforma al utilizar una versión de Java cuyos avances en la adaptación de código ejecutable en hardware de pocos recursos lo convierte en un robusto competidor.

10.2 Propuestas de mejora

En esta primera versión de la aplicación encontramos que a pesar de que cumple con el objetivo principal de transmisión y recepción de datos podemos incluir ciertas mejoras que contribuirían a la eficiencia de la misma, las cuales se detallarán a continuación.

En el binomio módem-contador podemos incluir que el maestro (en este caso el módem) controle más de un esclavo o contador gracias a que en Modbus se puede gestionar la comunicación de varios elementos de tipo esclavo, habilidad muy útil por ejemplo en el caso de incluir un módem con función de concentrador en un edificio en el que haya diversos suministros de electricidad cuyas medidas sea necesario enviar.

Para incluir esta funcionalidad sería preciso además disponer que un bus dado que el módem solo cuenta con 2 puertos serie o limitarnos simplemente a dos contadores.

Referente a la comunicación mediante mensajes HTTP podemos incluir varias mejoras como pueden ser, en primer lugar, enviar en los mensajes HTTP los datos incluidos en un objeto JSON que es más fácilmente parseable.

También y en torno a la seguridad podemos modificar la conexión HTTP por una HTTPS cuya clase está definida para JME también. De esta manera quedarían los datos del cuerpo del mensaje encriptados permitiendo un mayor grado de seguridad en la transferencia vía GPRS. Para llevar a cabo esta mejora sería necesario incluir certificados para la autenticación entre el cliente (módem) y el servidor.

En cuanto a la función de backup disponible en la versión del software actual también podemos añadir ciertas características que aporten mayor eficiencia a la aplicación, como pueden ser la comprobación del tamaño del fichero para que cuando este supere cierta cantidad de datos se avise al servidor de que las medidas almacenadas en el fichero deben ser recuperadas, o que periódicamente este fichero pueda ser enviado a través de FTP. Otra forma podría ser utilizar el método readFile para leer los datos del fichero y mandarlos vía HTTP sin necesidad de utilizar la transferencia del fichero.

Para la mejora del servidor socket TCP que se encuentra a la escucha, su funcionamiento actual solo implica que al recibir un inicio de conexión, se envíe un OK en respuesta y se envíen en cuanto sea posible los datos.

Si habilitamos la recepción en ese flujo de cierto tipo de mensajes podemos añadir funcionalidades extra tanto al programa como a la interacción en general. Por ejemplo podríamos recibir tres tipos de mensajes en los que el servidor central pueda pedir al módem una medida asíncrona, o en su defecto enviar en ese momento el fichero de backup vía FTP o por el contrario que se cambie el valor del timeout que utiliza el Timertask para que en lugar de 15 minutos que es el valor actual que implica que se haga la lectura periódica cada 15 minutos, ésta se realice cada hora por ejemplo. Para ello bastaría con recibir como información el nuevo valor de la variable measureIntervalTimeout que se fijaría en la ejecución del código de recibirse. Solo faltaría reiniciar el programa para que se genere el nuevo objeto Timertask con los nuevos valores.

Por último y en relación a la depuración de posibles fallos durante la ejecución del programa podemos incluir la escritura en un fichero de logs que se almacene en la memoria los distintos mensajes que actualmente contiene la aplicación y que durante la ejecución se muestran por la salida estándar. Y aún más útil si se imprimen en el fichero los valores devueltos en la captura de una excepción por el método “printStackTrace ()”.

De esta manera si durante la ejecución se captura alguna de las excepciones manejadas podremos conocer los detalles sin necesidad de depurar la aplicación en un dispositivo con pantalla que nos muestre la salida estándar, bastaría con recuperar el fichero de log vía FTP con la memoria. Así no solo descubriríamos la excepción que se está produciendo sino que además podríamos intentar solventar el problema que se está produciendo en una nueva versión de la aplicación que desplegaríamos vía OTAP.

ANEXOS

I. Códigos de ejemplo

HelloWorld.java

```
package src;

import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;

public class main extends MIDlet {

    // Constructor
    public main() {

    }

    protected void destroyApp(boolean arg0) throws MIDletStateChangeException
    {
        notifyDestroyed();
    }

    protected void pauseApp() {

    }

    protected void startApp() throws MIDletStateChangeException
    {
        // Printing the message in stdout
        System.out.println("Hello World");
        destroyApp(true);
    }
}
```

Thread.java

```
package src;

import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;

public class main extends MIDlet
{

    boolean runThreads = true;
    ThreadExample threadExampleObj;

    // Constructor
    public main() { }

    protected void destroyApp(boolean arg0)
            throws MIDletStateChangeException
    {
        try
        {
            // Join method waits for this thread to die
            threadExampleObj.join();
        }
        catch (Exception e)
        {
        }
        System.out.println("App Destroyed");
        notifyDestroyed();
    }

    protected void pauseApp() {}

    protected void startApp() throws MIDletStateChangeException
    {
        try
        {
            // Instantiate a new Thread Example Object
            threadExampleObj = new ThreadExample();
            threadExampleObj.start();
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
        destroyApp(true);
    }

    //Class that extends class Thread
    private class ThreadExample extends Thread
    {
        // Constructor
        public ThreadExample() {}

        public void run()
        {
            //Printing current index
            for (int i = 1; i <= 5; i++)
            {
                System.out.println("Current index is " + i);
```

```
                // Waits for a second
                try
                {
                    Thread.sleep(100);
                }
                catch(InterruptedException e)
                {
                    e.printStackTrace();
                }
            }
            System.out.println("Thread End");
        }
    }
}
```

Timer.java

```
package src;

import java.util.Timer;
import java.util.TimerTask;

import javax.microedition.midlet.*;

public class main extends MIDlet
{
    int counter=0;

    protected void startApp() throws MIDletStateChangeException
    {
        // Creating a Timer Object
        Timer timerMon = new Timer();
        // Creating a Monitor Object (extends timertask)
        Monitor mon = new Monitor();
        // Setting timertask parameters
        timerMon.schedule(mon,0,5000);
        destroyApp(true);
    }

    // Class Monitor which run method will be execute
    // when a new Monitor object is created and scheduled
    private class Monitor extends TimerTask
    {
        // When timeout occurs this code will be execute
        public void run()
        {
            counter++;
            System.out.println(counter);
        }

    }

    protected void pauseApp() {}

    protected void destroyApp(boolean arg0) throws MIDletStateChangeException
    {
        notifyDestroyed();
    }
}
```

II. Código implementado

```
/*
 * App Module Version 12
 *
 * COMPLETE VERSION APPLICATION
 *
 */

package src;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.DataInputStream;
import java.io.DataOutputStream;

import java.util.Timer;
import java.util.TimerTask;

import javax.microedition.io.Connector;
import javax.microedition.io.HttpConnection;
import javax.microedition.io.ServerSocketConnection;
import javax.microedition.io.SocketConnection;
import javax.microedition.io.StreamConnection;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;

import com.cinterion.io.ATCommand;
import com.cinterion.io.ATCommandListener;
import com.cinterion.io.ATCommandFailedException;
import com.cinterion.io.file.FileConnection;


/*
 * CLASS MAIN
 *
 * It contains all classes an methods which are
 * use for the application
 *
 */

public class main extends MIDlet
{

    // Value of the time in which the timeout occurs
    // 1 second = 1000
    private int measureIntervalTimeout = 900000;

    // This variable turns into true value when an asynchronous
    // request arrives from server
    private boolean pendingServerRequest = false;

    // This variable turns into true value when we can manage
    // the request from server
    private boolean canHandleServerRequest = false;

    // This variable represents the name of the available port
    // for the serial communication
    private String portAvailable = "";
```

```
// This is the URL of the HTTP server where we send the
// HTTP request
private String url = "http://52.4.102.93/";

// This string store the incoming data from serial port
private String stringReceivedDataFromSerialPort = "";

// It contains the parameters for the GPRS connection
private String connProfile = ";bearer_type=gprs;access_point=wlgdsp.com;" +
        "username=well;password=well;timeout=-1;";

// Stream Connection for the server socket
private ServerSocketConnection serverSocketConn = null;


private SocketThreadClass listenSocketThread = new SocketThreadClass();
private CheckOpenListenSocketClass isOpenSocketListenThread =
        new CheckOpenListenSocketClass();

Timer timer= null;


// Method startApp from MIDlet class
// When the app starts this method is called automatically
protected void startApp() throws MIDletStateChangeException
{
    try
    {
        System.out.println("\nSync time...\n");

        //Waiting for 3 seconds
        try
        {
            Thread.sleep(300);
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }

        // Calling the syncTime method to get the current time
        syncTime();
        System.out.println("\nClock Uptated\n");
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }

    try
    {
        System.out.println("\nChecking available ports...\n");

        // Calling the testAvailablePorts to get the available
        // ports names
        portAvailable = testAvailablePorts();
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }

    try
```

```
        {
            System.out.println("\nOpenning Server Socket...\n");

            // Openning the server socket connection
            serverSocketConn = (ServerSocketConnection)
                    Connector.open("socket://:23"+connProfile);
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }


        try
        {
            System.out.println("\nSetting Timertask...\n");
            TimertaskClass timertaskObj = new TimertaskClass();
            timer = new Timer();

            // Setting the timertask
            timer.scheduleAtFixedRate(timertaskObj, 0, measureIntervalTimeout);
        }
        catch (Exception e)
        {
            e.printStackTrace();
            destroyApp(true);
        }
    }


    // Method destroyApp from MIDlet class
    // When it is called this method destroy the app
    protected void destroyApp(boolean unconditional)
            throws MIDletStateChangeException
    {
        try
        {
            listenSocketThread.join();
            if (serverSocketConn != null)
                serverSocketConn.close();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        System.out.println("\nServer Socket closed. App destroyed.\n");
        notifyDestroyed();
    }


    // Method pauseApp from MIDlet class
    // When it is called pause the app
    protected void pauseApp() {}

    // Method syncTime created to get the current time
    // from network and set it in the MTX65i
    protected void syncTime ()
    {
        int quotes = 0;

        try
        {
```

```
            String dateString = "";

            // Required variables to listen and execute
            // AT Commands
            ATCommandListener LIS = new ATListener();
            ATCommand ATC = new ATCommand(false);
            ATC.addListener(LIS);

            // Sending AT^SIND Command to get the current time
            String resultSindCommand = ATC.send("AT^SIND=NITZ,1\r");
            try
            {
                Thread.sleep(100);
            }
            catch (InterruptedException e)
            {
                e.printStackTrace();
            }

            System.out.println("\nAT^SIND execution returns:"
                    +resultSindCommand+"\n");

            // Getting the date string
            for (int i = 0; i < resultSindCommand.length(); i++)
            {
                char character = resultSindCommand.charAt(i);
                if(character == '\"')
                {
                    if(quotes == 1)
                    {
                        quotes = 0;
                        dateString = dateString+character;
                    }
                    else
                        quotes = 1;
                }
                if(quotes == 1)
                {
                    dateString = dateString+character;
                }
            }

//          Sending AT^CCLK Command to set the current time
            String resultClockCommand = ATC.send("AT+CCLK="+dateString+"\r");

            try
            {
                Thread.sleep(100);
            }
            catch (InterruptedException e)
            {
                e.printStackTrace();
            }
            System.out.println("\nAT+CCLK execution returns: "
                    +resultClockCommand+"\n");
        }
        catch (IllegalStateException e)
        {
            e.printStackTrace();
        }
        catch (ATCommandFailedException e)
        {
            e.printStackTrace();
```

```
    }
}


// Method testAvailablePorts created to call
// getProperty method and read the available
// ports names to use in serial communication
protected String testAvailablePorts ()
{
    String portResult ="";
    try
    {
        // This method returns a string which
        // contains all available ports names
        String ports = System.getProperty("microedition.commports");
        int comma = ports.indexOf(',');
        if (comma > 0)
            portResult = ports.substring(0, comma);

        else
            portResult = ports;

        System.out.println("\nPort available is: "+portResult+"\n");
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
    return portResult;
}


// Method serialPortProtocol created to get
// the measures from the serial communication
protected String serialPortProtocol ()
{
    System.out.println("\nReading from Serial Port...\n");

    String stringReadingFromSerialPort = "";
    String bufferOut = "";

    // Streams for input and output data
    InputStream in_stream_serial = null;
    OutputStream out_stream_serial = null;
    StreamConnection streamConn = null;

    bufferOut = ":01MRSendMeMeasurementData/n";

    try
    {
        if(streamConn != null)
            streamConn.close();

        streamConn = (StreamConnection)Connector.open("comm:"+
                portAvailable+";baudrate=115200;blocking=off");

    }
    catch(IOException e)
    {
        e.printStackTrace();
    }
```

```
        try
        {
            System.out.println("\nOpenning streams for serial communication\n");
            if (in_stream_serial == null)
                in_stream_serial  = streamConn.openInputStream();
            if (out_stream_serial == null)
                out_stream_serial = streamConn.openOutputStream();
            out_stream_serial.write(bufferOut.getBytes());

            int currentCharacter = -1;
            while(currentCharacter != 13)
            {
                currentCharacter = in_stream_serial.read();
                if (currentCharacter != -1)
                    System.out.print((char)currentCharacter);

                if((currentCharacter != -1) && (currentCharacter != 13))
                    stringReadingFromSerialPort = stringReadingFromSerialPort
                            + (char)currentCharacter;
            }
            System.out.println("\nModbus data read from serial port is: "
                    +stringReadingFromSerialPort+"\n");

            if(in_stream_serial != null)
                in_stream_serial.close();
            if(out_stream_serial != null)
                out_stream_serial.close();

        }
        catch (IOException e)
        {
            e.printStackTrace();
        }

        // In any case streams are closed
        finally
        {
            try
            {
                if(in_stream_serial != null)
                    in_stream_serial.close();
                if(out_stream_serial != null)
                    out_stream_serial.close();
                if( streamConn != null)
                    streamConn.close();
            }
            catch(IOException e)
            {
                e.printStackTrace();
            }
        }

        // This method returns the read data from
        // serial port
        return stringReadingFromSerialPort;
    }


    // Method HTTPRequest created send HTTP request
    // message with the measuring data
    protected void HTTPRequest (String httpToSendString)
    {
        System.out.println("\nInit HTTP Request to send data...\n");
```

```
int readChar = 0;
int respCode = 0;
String echoHTTPResponse = "";
OutputStream out_stream_http = null;
DataInputStream in_stream_http = null;
HttpConnection httpConn = null;

try
{
    // Openning HTTP Connection to send the message
    if (httpConn == null)
        httpConn = (HttpConnection)Connector.open(url + connProfile);
    httpConn.setRequestMethod(HttpConnection.POST);
    httpConn.setRequestProperty("User-Agent",
            "Profile/MIDP-1.0 Confirguration/CLDC-1.0");
    httpConn.setRequestProperty("Content-Type",
            "application/x-www-form-urlencoded");
    httpConn.setRequestProperty("Content-Length",
            Integer.toString(httpToSendString.length()));

    try
    {
        if (out_stream_http == null)
            out_stream_http = httpConn.openOutputStream();
        out_stream_http.write (httpToSendString.getBytes());
    }
    catch (Exception e)
    {
        System.out.println("Cannot write");
        e.printStackTrace();
    }
    System.out.println("\nWaiting for echo response from server ...\n");
    //Getting the response...
    respCode = httpConn.getResponseCode();

    //If there is no error with the data received ...
    if (respCode == HttpConnection.HTTP_OK)
    {
        if (in_stream_http == null)
            in_stream_http = httpConn.openDataInputStream();

        // Reading HTTP Response while stream is not emmpty
        while ((readChar = in_stream_http.read()) != -1)
        {
            echoHTTPResponse = echoHTTPResponse+ (char)readChar;
        }
        System.out.println("\nHTTP response echo is: "
                + echoHTTPResponse+"\n");
    }
    else
    {
        System.out.println("Not 200 OK. Needs debug. Response code: "
                + respCode);
    }

}
catch (IOException e){
    e.printStackTrace();
}

finally
{
```

```
            //Closing streams
            try
            {
                System.out.println("Closing streams...");
                if(in_stream_http!= null)
                    in_stream_http.close();
                if(out_stream_http != null)
                    out_stream_http.close();
                if(httpConn != null)
                    httpConn.close();
            }
            catch (IOException e)
            {
                e.printStackTrace();
            }
        }

        // Checking echo. If there is an error writeFile method is called
        if ((respCode != HttpConnection.HTTP_OK) ||
                !(httpToSendString.equals(echoHTTPResponse)))
        {
            System.out.println("\nErrors sending data. " +
                    "Storing data in backup file...\n");

            try
            {
                writeFile(httpToSendString);
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }

        }
    }


    // Method readFile created to read from a backup file
    protected void readFile(String data) throws MIDletStateChangeException
    {
        String readData = "";
        try
        {
            System.out.println("\nReading data to the file...\n");
            FileConnection fileConn = (FileConnection)
                    Connector.open("file:///a:/backup.txt");

            // If the file does not exist it is created
            // and we can read data
            if (fileConn.exists())
            {
                InputStream in_stream_file =
                    fileConn.openDataInputStream();

                if (in_stream_file.available() > -1)
                {
                    in_stream_file.read(readData.getBytes(), 0,
                            in_stream_file.available());
                }
                data = String.valueOf(readData.getBytes());

                in_stream_file.close();
                fileConn.close();
```

```
            }
            else
                System.out.println("\nFile does not exist.\n");
        }

        catch (IOException e)
        {
            e.printStackTrace();
        }
    }

    // Method writeFile created to write in a backup file
    protected void writeFile(String data) throws MIDletStateChangeException
    {
        System.out.println("\nWriting data to the file...\n");
        try
        {
            // Openning the file descriptor to write
            FileConnection fileConn = (FileConnection)
                    Connector.open("file:///a:/backup.txt");
            long sizeFile = 0;

            if (!fileConn.exists())
            {
                fileConn.create();
            }
            else
            {
                sizeFile = fileConn.fileSize();
                if(sizeFile == -1)
                    System.out.println("\nFile not accesible.\n");
            }

            // If file contains something we append the new data at the end
            OutputStream out_stream_file = fileConn.openOutputStream(sizeFile);
            data=data+"\n";
            out_stream_file.write(data.getBytes());
            out_stream_file.flush();
            out_stream_file.close();

            fileConn.close();
        }

        catch (IOException e)
        {
            e.printStackTrace();
        }
    }


    /*
     * CLASS ATLISTENER
     *
     * It is necessary to enable AT Command
     * execution from the app
     *
     */
    class ATListener implements ATCommandListener
    {
        public void ATEvent(String Event)
        {
            System.out.println("Evento: "+ Event);
```

```
    }

    // This methods are mandatory for the
    // class ATCommandListener
    public void RINGChanged(boolean Event) {}
    public void DCDChanged(boolean Event) {}
    public void DSRChanged(boolean Event) {}
    public void CONNChanged(boolean Event) {}

}



/*
 * CLASS SOCKETTHREADCLASS
 *
 * It manages the asynchronous request from
 * server by starting a connection socket
 * if a request is received
 *
 */

private class SocketThreadClass extends Thread
{
    SocketConnection socketConn = null;
    DataInputStream in_stream_server = null;
    DataOutputStream out_stream_server = null;

    String result = "";
    String responseServerRequestOK = "200 OK";

    // Constructor of SocketThreadClass
    public SocketThreadClass()
    {
        try
        {
            if (in_stream_server != null)
                in_stream_server.close();
            if (out_stream_server != null)
                out_stream_server.close();
            if (socketConn != null)
                socketConn.close();
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }

    public void run()
    {
        try
        {
            // Closing the streams
            try
            {
                if (in_stream_server != null)
                    in_stream_server.close();
                if (out_stream_server != null)
                    out_stream_server.close();
                if (socketConn != null)
                    socketConn.close();
            }
```

```
        catch (IOException e)
        {
            e.printStackTrace();
        }

        System.out.println("\nServer socket waiting " +
                "for an asynchronous data request...\n");

        // This line blocks the thread while no incoming
        // connection is received
        socketConn = (SocketConnection)
                serverSocketConn.acceptAndOpen();

        System.out.println("\nAsynchronous data " +
                "request received...\n");

        // When a request is received streams are opened
        out_stream_server =
                socketConn.openDataOutputStream();
        out_stream_server.write(responseServerRequestOK.getBytes());
        socketConn.close();

        // Setting pendingServerRequest to true
        // The main thread will check this variable
        // and will not use serialPortProtocol
        // or HTTPResquest methods
        pendingServerRequest = true;

        // Until canHandleServerRequest is not true
        // we cannot use the serialPortProtocol
        // or HTTPResquest methods
        while (!canHandleServerRequest)
        {
            try
            {
                Thread.sleep(1000);
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
        }
        try
        {
            stringReceivedDataFromSerialPort = serialPortProtocol();
            HTTPRequest(stringReceivedDataFromSerialPort);
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }

        if (in_stream_server != null)
            in_stream_server.close();
        if (out_stream_server != null)
            out_stream_server.close();

        System.out.println("\nAttended request and data sent\n");
        pendingServerRequest = false;
    }
    catch (IOException e)
    {
```

```
            e.printStackTrace();
        }
    }
}


/*
 * CLASS CHECKOPENLISTENSOCKETCLASS
 *
 * It keeps alive the socket thread in
 * order to listen any incoming request
 * from server
 *
 */
private class CheckOpenListenSocketClass extends Thread
{
    public CheckOpenListenSocketClass () {}

    public void run ()
    {
        while (true)
        {
            try
            {
                Thread.sleep(1000);
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }

            // If Socket Thread is not alive
            // we start again this thread to
            // listen incoming connections
            if (!listenSocketThread.isAlive())
            {
                try
                {
                    Thread.sleep(1000);
                }
                catch (Exception e)
                {
                    e.printStackTrace();
                }
                listenSocketThread = new SocketThreadClass();
                listenSocketThread.start();
            }
            else
            {
                try
                {
                    Thread.sleep(1000);
                }
                catch (Exception e)
                {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

```
/*
 * CLASS TIMERTASKCLASS
 *
 * Its run method do the synchronous measuring
 * request to the smart meter
 *
 */
private class TimertaskClass extends TimerTask
{
    public TimertaskClass() {}

    public void run()
    {
        // Checking if there is a pending server
        // asyncrhonous request to wait until this
        // request is attended
        if (pendingServerRequest)
        {
            try
            {
                canHandleServerRequest = true;
                while (pendingServerRequest)
                {
                    try
                    {
                        Thread.sleep(1000);
                    }
                    catch (Exception e)
                    {
                        e.printStackTrace();
                    }
                }
                canHandleServerRequest = false;
                stringReceivedDataFromSerialPort = serialPortProtocol();
                HTTPRequest(stringReceivedDataFromSerialPort);
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
        }

        // If there is no pending server request
        // we could get the measuring data without
        // wait
        else
        {
            try
            {
                canHandleServerRequest = false;
                stringReceivedDataFromSerialPort = serialPortProtocol();
                HTTPRequest(stringReceivedDataFromSerialPort);
                try
                {
                    if (!listenSocketThread.isAlive())
                    {
                        try
                        {
                            Thread.sleep(1000);
                        }
                        catch (Exception e)
                        {
```

```
                                e.printStackTrace();
                            }
                            listenSocketThread = new SocketThreadClass();
                            listenSocketThread.start();
                        }
                    }
                    catch(Exception e)
                    {
                        System.out.println(e);
                    }
                }
                catch (Exception e)
                {
                    e.printStackTrace();
                }

            }

            // When the synchronous measuring is done
            // we set canHandletServerRequest to true
            // for enable the asynchronous manage until
            // the timeout occurs
            canHandleServerRequest = true;
            try
            {
                // Checking if the open socket test is alive
                if (!isOpenSocketListenThread.isAlive()){

                    try
                    {
                        Thread.sleep(1000);
                    }
                    catch (Exception e)
                    {
                        e.printStackTrace();
                    }
                    isOpenSocketListenThread = new CheckOpenListenSocketClass();
                    isOpenSocketListenThread.start();
                }
            }
            catch(Exception e)
            {
                System.out.println(e);
            }
        }
    }
}
```

III. Relación de Comandos AT relevantes [69]

2.1 AT&F Reset AT Command Settings to Factory Default Values

AT&F resets AT command settings to their factory default values.
Every ongoing or incoming call will be terminated.
All defined GPRS contexts which are not activated or not online will be undefined if the non volatile storage of AT+CGDCONT settings is not activated by AT^SCFG parameter "GPRS/PersistentContexts".
For a list of affected parameters refer to Section 23.6, Factory Default Settings Restorable with AT&F.

Syntax

Exec Command
AT&F[<value>]
Response(s)
OK

PIN	ASC0	ASC1	USB	MUX1	MUX2	MUX3	Charge	✈	Last
-	+	+	+	+	+	+	-	+	-

Reference(s)
V.250

Parameter Description

<value>(num)	
[0]	Reset parameters in Section 23.6, Factory Default Settings Restorable with AT&F to their factory default values.

2.2 AT&V Display current configuration

AT&V returns the current parameter setting. The configuration varies depending on whether or not PIN authentication has been done.

Syntax

Exec Command
AT&V[<value>]
Response(s)
ACTIVE PROFILE: ... (see Section 2.2.1, AT&V responses) OK

PIN	ASC0	ASC1	USB	MUX1	MUX2	MUX3	Charge	✈	Last
-	+	±	+	+	±	±	-	+	-

Reference(s)
V.250

Parameter Description

<value>(num)	
[0]	Profile number

Notes

- The value of \Q (flow control) is also determined by the AT+IFC command. In case the value set by AT+IFC cannot be represented by a \Q equivalent, \Q255 will be displayed.
- The parameters of AT^SMGO can only be displayed after the SMS data from the SIM have been read successfully for the first time. Reading starts after successful SIM authentication has been performed, and may take up to 30 seconds depending on the SIM used. While the read process is in progress, an attempt to read the parameter will result in empty values.
- The parameter of AT+CSDH will only be displayed in SMS PDU mode, see AT+CMGF.

2.2.1 AT&V responses

The following tables show four different kinds of responses depending on whether or not the PIN is entered and whether or not the Multiplex mode is enabled (see AT+CMUX).

Table 2.1: Current configuration on ASC0 / MUX channel 1 / USB (example)

PIN authentication done	No PIN authentication
ACTIVE PROFILE: E1 Q0 V1 X4 &C1 &D2 &S0 \Q0 \V1 S0:000 S3:013 S4:010 S5:008 S6:000 S7:060 S8:000 S10:002 S18:000 +CBST: 7,0,1 +CRLP: 61,61,78,6 +CR: 0 +FCLASS: 0 +CRC: 0 +CMGF: 1 +CSDH: 0 +CNMI: 0,0,0,0,1 +ICF: 3 +IFC: 0,0 +ILRR: 0 +IPR: 115200 +CMEE: 2 ^SMGO: 0,0 +CSMS: 0,1,1,1 ^SACM: 0,"000000","000000" ^SLCC: 0 ^SCKS: 0,1 ^SSET: 0 +CREG: 0,1 +CLIP: 0,2 +CAOC: 0 +COPS: 0,0,"operator" +CGSMS: 3 OK	ACTIVE PROFILE: E1 Q0 V1 X4 &C1 &D2 &S0 \Q0 \V1 S0:000 S3:013 S4:010 S5:008 S6:000 S7:060 S8:000 S10:002 S18:000 +CBST: 7,0,1 +CRLP: 61,61,78,6 +CR: 0 +FCLASS: 0 +ICF: 3 +IFC: 0,0 +ILRR: 0 +IPR: 115200 +CMEE: 2 ^SCKS: 0,1 ^SSET: 0 OK

2.3 AT&W Store AT Command Settings to User Defined Profile

AT&W stores the current AT command settings to a user defined profile in non-volatile memory of TC65i. The AT command settings will automatically be restored from the user defined profile during power-up or if ATZ is used. AT&F restores AT command factory default settings. Hence, until first use of AT&W, ATZ works as AT&F.
A list of parameters stored to the user profile can be found at Section 23.5, AT Command Settings storable with AT&W.

Syntax

Exec Command

AT&W[<value>]

Response(s)

OK
ERROR
+CME ERROR: <err>

PIN	ASC0	ASC1	USB	MUX1	MUX2	MUX3	Charge	✈	Last
-	+	+	+	+	+	+	-	+	-

Reference(s)

V.250

Parameter Description

<value>(num)

[0]	User Profile Number

2.8 ATZ Restore AT Command Settings from User Defined Profile

First ATZ resets the AT command settings to their factory default values, similar to AT&F. Afterwards the AT command settings are restored from a user defined profile in non-volatile memory of TC65i, if one was stored with AT&W before. Any additional AT command on the same command line may be ignored. A delay of 300 ms is required before next AT command is sent.
If a connection is in progress, it will be terminated.
All defined GPRS contexts which are not activated or not online will be undefined if the non volatile storage of CGDCONT settings is not activated by the AT^SCFG parameter "GPRS/PersistentContexts" (see AT+CGDCONT).

Syntax

Exec Command
ATZ[<value>]
Response(s)
OK

PIN	ASC0	ASC1	USB	MUX1	MUX2	MUX3	Charge	✈	Last
-	+	+	+	+	+	+	-	+	-

Reference(s)
V.250

Parameter Description

<value>(num)

[0] User Profile Number

2.9 AT+CFUN Functionality Level

AT+CFUN controls the TC65i's functionality level. It can be used to reset the ME, to choose one of the power save (SLEEP) modes or to return to full functionality.
Intended for power saving, SLEEP mode usage reduces the functionality of the ME to a minimum and thus minimizes the current consumption. Further information, particularly power supply ratings during the various operating modes and the timing of UART signals in SLEEP mode can be found in the "TC65i Hardware Interface Description, Version 02.004".
The selected <fun> parameter is global for all instances and needs to be set only on one instance. Nevertheless, power save management differs significantly for the two interface types UART and USB:

- For UART-only operation it is sufficient to select one of the <fun> levels and to activate hardware flow control both on the ME (AT\Q3) and on each application connected to the UART (ASC0 and/or ASC1).
- If the USB interface is involved select one of the <fun> levels (unless done so on another instance) and either disconnect the USB interface or set the USB interface into Suspend mode. Otherwise, if you change only the AT+CFUN parameters, the selected <fun> level will be accepted but the active USB keeps the module alive and thus prevents power saving until USB is disconnected. Vice versa, if the module is in SLEEP mode restarting the USB will cause the module to stop power saving although the selected <fun> level does not change.
- Power save management is handled independently on both interface types. CTS state transitions described below for the UART are not effective for USB. On the USB interface the power saving process is determined by USB Suspend state and USB Remote Wakeup mechanisms specified in the Universal Serial Bus Revision 2.0 Specification [47]. See also "Application Note 32: Integrating USB into GSM Applications".
- To check whether or not power saving is effective you may monitor the CTS line of the UART or the Status LED connected to the SYNC pin. In the case of USB the power saving status is indicated only by the Status LED. See AT^SSYNC for more information regarding the LED. See also note below.
- A wakeup event signalled only on the UART (e.g. URCs) will not wake up the USB instance, and vice versa, a wakeup event signalled only on the USB interface will not wake up the UART instances. Therefore, keep in mind that most of the URCs can be activated separately for each AT command instance.

Power save (SLEEP) modes fall in two categories:

- NON-CYCLIC SLEEP mode selectable with <fun>=0 (not recommended on USB interface)
- and CYCLIC SLEEP modes selectable with <fun>= 7 or 9.

NON-CYCLIC SLEEP mode:

- If set on the UART (ASC0 and/or ASC1) the AT command interfaces will be permanently blocked.
- If the USB interface is connected and in Suspend state NON-CYCLIC SLEEP mode is not recommended because some wakeup events on the USB interface would cause the ME to exit SLEEP mode even though USB has been set to Suspend state.

CYCLIC SLEEP mode:

- On the UART, the benefit of CYCLIC SLEEP mode is that the AT interface remains accessible and that, in intermittent wakeup periods, characters can be sent or received without terminating the selected mode. This allows the ME to wake up for the duration of an event and, afterwards, to resume power saving. By setting/resetting the CTS signal the ME indicates to the application whether or not the UART is active. A summary of all SLEEP modes and the different ways of waking up the module on the UART can be found in Section 2.9.1, Wake up the ME from SLEEP mode (for UART only).
- On the USB interface, characters can be sent or received any time. The power saving process is handled by the Suspend state mechanisms. If a wakeup event occurs (e.g. URC, RING, data) the USB Remote Wakeup mechanism can be used as specified in the Universal Serial Bus Revision 2.0 Specification [47].
- In all CYCLIC SLEEP modes, you can enter <fun>=1 to permanently wake up TC65i and take it back to full functionality.

SLEEP mode management if Java is started:
The Java Virtual Machine remains active, but also enters the SLEEP mode. AT commands can be sent from the Java application to the serial interface, no matter which SLEEP mode was selected. This allows you to control the ME even if it is in NON-CYCLIC SLEEP mode.

AT+CFUN test command returns the values of the supported parameters.
AT+CFUN read command returns the current functionality value.
AT+CFUN write command can be used to reset the ME, to choose one of the SLEEP modes or to return to full functionality.

Syntax

Test Command

```
AT+CFUN=?
```

Response(s)

```
+CFUN: (list of supported <fun>s) , (list of supported <rst>s)
OK
```

Read Command

```
AT+CFUN?
```

Response(s)

```
+CFUN: <fun>
OK
```

Write Command

```
AT+CFUN=[<fun>[, <rst>]]
```

Response(s)

```
OK
ERROR
+CME ERROR: <err>
```

PIN	ASC0	ASC1	USB	MUX1	MUX2	MUX3	Charge	✈	Last
-	+	+	+	±	±	±	-	+	-

Reference(s)

3GPP TS 27.007 [42]

Unsolicited Result Codes

URC 1

^SYSSTART

Indicates that the ME has been started and is ready to operate. If autobauding is active (AT+IPR=0) the URC is not generated.

If the ME is configured to enter the Airplane mode after restart or reset the following URC is received on boot-up: "^SYSSTART AIRPLANE MODE". In this mode, only the AT commands listed in Section 23.4, Availability of AT Commands Depending on Operating Mode of ME can be used. For details please refer to the AT^SCFG command, parameter <mapos>.

2.12 AT+CMEE Error Message Format

AT+CMEE controls the format of error result codes that indicates errors related to TC65i functionality. Format can be selected between plain "ERROR" output, error numbers or verbose "+CME ERROR: <err>" and "+CMS ERROR: <err>" messages.
Possible error result codes are listed in Table 2.4, General "CME ERROR" Codes (3GPP TS 27.007), Table 2.5, General "CME ERROR" Codes (proprietary), Table 2.6, GPRS related "CME ERROR" Codes (3GPP TS 27.007), Table 2.9, Java related "CME ERROR" Codes (proprietary) and Table 2.8, SMS related "CMS ERROR" Codes (3GPP TS 27.005).
In multiplex mode (refer AT+CMUX) the setting applies only to the logical channel where selected. The setting on the other channels may differ.

Syntax

Test Command

```
AT+CMEE=?
```

Response(s)

```
+CMEE: (list of supported<errMode>s)
OK
```

Read Command

```
AT+CMEE?
```

Response(s)

```
+CMEE: <errMode>
OK
```

Write Command

```
AT+CMEE=<errMode>
```

Response(s)

```
OK
ERROR
+CME ERROR: <err>
```

Parameter Description

<errMode>(num)(&W)(&V)	
0(&F)(D)	Disable result code, i.e. only "ERROR" will be displayed.
1	Enable error result code with numeric values.
2	Enable error result code with verbose (string) values.

Example

To obtain enhanced error messages it is recommended to choose <errMode>=2.

```
AT+CMEE=2
OK
```

2.14 AT^SCFG Extended Configuration Settings

AT^SCFG can be used to query and configure various settings of the TC65i.
AT^SCFG read command returns a list of all supported parameters and their current values.
AT^SCFG write command queries a configuration parameter (if no value is entered) or sets its value(s).
Input of parameter names is always coded in GSM character set, parameter values are expected to be given as specified via AT+CSCS.

Syntax

Test Command

AT^SCFG=?

Response(s)

^SCFG: "AutoExec" , (list of supported <AutoExecCmd>), (list of supported <AutoExecType>), (list of supported <AutoExecIndex>), (list of supported <AutoExecMode>), (max. string length of <AutoExecATC>), (time range of <AutoExecPeriod>)
^SCFG: "Call/Ecall/T3242", (list of supported s)
^SCFG: "Call/Ecall/T3243", (list of supported s)
^SCFG: "Call/ECC", (list of supported <ecc>s)
^SCFG: "GPRS/ATS0/withAttach", (list of supported <gs0aa>s)
^SCFG: "GPRS/AutoAttach", (list of supported <gaa>s)
^SCFG: "GPRS/PersistentContexts", (list of supported <gpc>s)
^SCFG: "GPRS/RingOnIncomingData", (list of supported <groid>s)
^SCFG: "MEopMode/Airplane", (list of supported <map>s)
^SCFG: "MEopMode/Airplane/OnStart", (list of supported <mapos>s)
^SCFG: "MEopMode/CregRoam", (list of supported <mrs>s)
^SCFG: "MEShutdown/OnIgnition", (list of supported <msi>s)
^SCFG: "PowerSaver/Mode9/Timeout", (list of supported <psm9to>s)
^SCFG: "Radio/Band", (list of supported <rbp>s), (list of supported <rba>s)
^SCFG: "Radio/CNS", (list of supported <cns>s)
^SCFG: "Radio/OutputPowerReduction", (list of supported <ropr>s)
^SCFG: "SAT/GTP", (list of supported <gtp>s)
^SCFG: "Serial/Ifc", (list of supported <ifcMode>s)
^SCFG: "Serial/USB/DDD" , (list of supported <deviceDescr>s) , (list of supported <descrIndex>s) , (max. string length of <langId>) , (max. string length of <vendorId>) , (max. string length of <productId>) , (max. string length of <manufacturer>) , (max. string length of <product>) , (max. string length of <serialNo>)
^SCFG: "Tcp/BufSize", (list of supported <tcpBufSize>)
^SCFG: "Tcp/IRT", (list of supported <tcpIrt>)
^SCFG: "Tcp/MR", (list of supported <tcpMr>)
^SCFG: "Tcp/OT", (list of supported <tcpOt>)
^SCFG: "Tcp/SAck", (list of supported <tcpSack>)
^SCFG: "Tcp/TTcp", (list of supported <tcpTtcp>)
^SCFG: "Tcp/WithURCs", (list of supported <tcpWithUrc>)
^SCFG: "Trace/Syslog/OTAP", (list of supported <otapTracer>)
^SCFG: "URC/CallStatus/CIEV", (list of supported <succ>s)
^SCFG: "URC/CallStatus/SLCC", (list of supported <sucs>s)
^SCFG: "URC/Datamode/Ringline", (list of supported <udri>s)
^SCFG: "URC/Ringline", (list of supported <uri>s)
^SCFG: "URC/Ringline/ActiveTime", (list of supported <urat>s)
^SCFG: "Userware/Autostart", (list of supported <ua>s)
^SCFG: "Userware/Autostart/AppName", (max. string lengths of <uaa>)
^SCFG: "Userware/Autostart/Delay", (list of supported <uad>s)
^SCFG: "Userware/Passwd", (max. string length of <upwd>)
^SCFG: "Userware/DebugInterface", (<udbgif1>), (<udbgif2>), (<udbgif3>)
^SCFG: "Userware/Mode", (list of supported <umode>), (length of <uurl>), (range of supported <uport>values)

Test Command (Continued)

AT^SCFG=?

Response(s)

^SCFG: "Userware/Stdout", (list of supported <if>), (list of supported <intvalue>), (<filename>), (list of supported <logmode>)
^SCFG: "Userware/Watchdog", (list of supported <wd>)
OK

Read Command

AT^SCFG?

Response(s)

^SCFG: "AutoExec" , <AutoExecCmd>, <AutoExecType>, <AutoExecIndex>, <AutoExecMode>, <AutoExecATC>[, <AutoExecPeriod>, <AutoExecPeriodTimeLeft>]
^SCFG: "Call/ECC", <ecc>
^SCFG: "GPRS/ATS0/withAttach", <gs0aa>
^SCFG: "GPRS/AutoAttach", <gaa>
^SCFG: "GPRS/PersistentContexts", <gpc>
^SCFG: "GPRS/RingOnIncomingData", <groid>
^SCFG: "MEopMode/Airplane", <map>
^SCFG: "MEopMode/Airplane/OnStart", <mapos>
^SCFG: "MEopMode/CregRoam", <mrs>
^SCFG: "MEShutdown/OnIgnition", <msi>
^SCFG: "PowerSaver/Mode9/Timeout", <psm9to>
^SCFG: "Radio/Band", <rbc>, <rba>
^SCFG: "Radio/CNS", <cns>
^SCFG: "Radio/OutputPowerReduction", <ropr>
^SCFG: "SAT/GTP", <gtp>
^SCFG: "Serial/Ifc", <ifcMode>
^SCFG: "Serial/USB/DDD" , <deviceDescr>, <descrIndex>, <langId>, <vendorId>, <productId>, <manufacturer>, <product>, <serialNo>
^SCFG: "Tcp/BufSize", <tcpBufSize>
^SCFG: "Tcp/IRT", <tcpIrt>
^SCFG: "Tcp/MR", <tcpMr>
^SCFG: "Tcp/OT", <tcpOt>
^SCFG: "Tcp/SAck", <tcpSack>
^SCFG: "Tcp/TTcp", <tcpTtcp>
^SCFG: "Tcp/WithURCs", <tcpWithUrc>
^SCFG: "Trace/Syslog/OTAP", <otapTracer>
^SCFG: "URC/CallStatus/CIEV", <succ>
^SCFG: "URC/CallStatus/SLCC", <sucs>
^SCFG: "URC/Datamode/Ringline", <udri>
^SCFG: "URC/Ringline", <uri>
^SCFG: "URC/Ringline/ActiveTime", <urat>
^SCFG: "Userware/Autostart", <ua>
^SCFG: "Userware/Autostart/AppName", <uaa>
^SCFG: "Userware/Autostart/Delay", <uad>
^SCFG: "Userware/DebugInterface", <udbgif1>, <udbgif2>, <udbgif3>
^SCFG: "Userware/Mode"
^SCFG: "Userware/Stdout", <if>[, <intvalue>[, <filename>[, <logmode>]]]
^SCFG: "Userware/Watchdog", <wd>
OK

Write Command

Configure Audio Loop:

AT^SCFG="Audio/Loop"[, <al>]

Response(s)

^SCFG: "Audio/Loop", <al>
OK
ERROR
+CME ERROR: <err>

Write Command

Automatic AT command execution

AT^SCFG="AutoExec" , <AutoExecCmd>, <AutoExecType>, <AutoExecIndex>[, <AutoExecMode>, <AutoExecATC>[, <AutoExecPeriod>]]

Response(s)

^SCFG: "AutoExec" , <AutoExecCmd>, <AutoExecType>, <AutoExecIndex>, <AutoExecMode>, <AutoExecATC>[, <AutoExecPeriod>, <AutoExecPeriodTimeLeft>]
OK
ERROR
+CME ERROR: <err>

Write Command

Query/Configure Emergency numbers for SIM without ECC field

AT^SCFG="Call/ECC"[, <ecc>]

Response(s)

^SCFG: "Call/ECC", <ecc>
OK
ERROR
+CME ERROR: <err>

Write Command

GPRS ATS0 with automatic attach

AT^SCFG="GPRS/ATS0/withAttach"[, <gs0aa>]

Response(s)

^SCFG: "GPRS/ATS0/withAttach", <gs0aa>
OK
ERROR
+CME ERROR: <err>

Write Command

Automatic GPRS attach

AT^SCFG="GPRS/AutoAttach"[, <gaa>]

Response(s)

^SCFG: "GPRS/AutoAttach", <gaa>
OK
ERROR
+CME ERROR: <err>

Write Command

Persistent GPRS contexts

AT^SCFG="GPRS/PersistentContexts"[, <gpc>]

Response(s)

^SCFG: "GPRS/PersistentContexts", <gpc>
OK

Write Command (Continued)

Persistent GPRS contexts

AT^SCFG="GPRS/PersistentContexts"[, <gpc>]

Response(s)

ERROR
+CME ERROR: <err>

Write Command

Ring on incoming GPRS IP data packets

AT^SCFG="GPRS/RingOnIncomingData"[, <groid>]

Response(s)

^SCFG: "GPRS/RingOnIncomingData", <groid>
OK
ERROR
+CME ERROR: <err>

Write Command

Enable/disable Airplane mode during operation

AT^SCFG="MEopMode/Airplane"[, <map>]

Response(s)

^SCFG: "MEopMode/Airplane", <map>
OK
ERROR
+CME ERROR: <err>

Write Command

Airplane mode upon ME restart

AT^SCFG="MEopMode/Airplane/OnStart"[, <mapos>]

Response(s)

^SCFG: "MEopMode/Airplane/OnStart", <mapos>
OK
ERROR
+CME ERROR: <err>

Write Command

AT^SCFG="MEopMode/CregRoam"[, <mrs>]

Response(s)

^SCFG: "MEopMode/CregRoam", <mrs>
OK
ERROR
+CME ERROR: <err>

Write Command

Enable/disable shutdown by ignition line.

AT^SCFG="MEShutdown/OnIgnition"[, <msi>]

Response(s)

^SCFG: "MEShutdown/OnIgnition", <msi>
OK
ERROR
+CME ERROR: <err>

Write Command

Query/Set timeout value for power saving mode 9

AT^SCFG="PowerSaver/Mode9/Timeout"[, <psm9to>]

Response(s)

^SCFG: "PowerSaver/Mode9/Timeout", <psm9to>
OK
ERROR
+CME ERROR: <err>

Write Command

Radio band selection

AT^SCFG="Radio/Band"[, <rbp>][, <rba>]

Response(s)

^SCFG: "Radio/Band", <rbp>, <rba>
OK
ERROR
+CME ERROR: <err>

Write Command

Query/Enable/Disable Continuous Network Search

AT^SCFG="Radio/CNS"[, <cns>]

Response(s)

^SCFG: "Radio/CNS", <cns>
OK
ERROR
+CME ERROR: <err>

Write Command

Radio output power reduction

AT^SCFG="Radio/OutputPowerReduction"[, <ropr>]

Response(s)

^SCFG: "Radio/OutputPowerReduction", <ropr>
OK
ERROR
+CME ERROR: <err>

Write Command

SAT GSM Terminal Profile

AT^SCFG="SAT/GTP"[, <gtp>]

Response(s)

^SCFG: "SAT/GTP", <gtp>
OK
ERROR
+CME ERROR: <err>

Write Command

Configuration of TCP parameter 'BufSize':

AT^SCFG="Tcp/BufSize"[, <tcpBufSize>]

Response(s)

^SCFG: "Tcp/BufSize", <tcpBufSize>
OK
ERROR
+CME ERROR: <err>

Write Command

Configuration of TCP parameter 'InitialRetransmissionTimeout':

AT^SCFG="Tcp/IRT"[, <tcpIrt>]

Response(s)

^SCFG: "Tcp/IRT", <tcpIrt>
OK
ERROR
+CME ERROR: <err>

Write Command

Configuration of TCP parameter 'MaxRetransmissions':

AT^SCFG="Tcp/MR"[, <tcpMr>]

Response(s)

^SCFG: "Tcp/MR", <tcpMr>
OK
ERROR
+CME ERROR: <err>

Write Command

Configuration of TCP parameter 'OverallTimeout':

AT^SCFG="Tcp/OT"[, <tcpOt>]

Response(s)

^SCFG: "Tcp/OT", <tcpOt>
OK
ERROR
+CME ERROR: <err>

Write Command

Configuration of TCP parameter 'Selective Acknowledge':

AT^SCFG="Tcp/SAck"[, <tcpSack>]

Response(s)

^SCFG: "Tcp/SAck", <tcpSack>
OK
ERROR
+CME ERROR: <err>

Write Command

Configuration of Internet Service URCs:

AT^SCFG="Tcp/WithURCs"[, <tcpWithUrc>]

Response(s)

^SCFG: "Tcp/WithURCs", <tcpWithUrc>
OK
ERROR
+CME ERROR: <err>

Write Command

Tracing:

AT^SCFG="Trace/Syslog/OTAP"[, <otapTracer>]

Response(s)

^SCFG: "Trace/Syslog/OTAP", <otapTracer>
SYSLOG ENABLED
ERROR
+CME ERROR: <err>

Write Command

Configure transparent communication link (tunnel) between different USB / UART or Mux ports.

AT^SCFG="Serial/Ifc"[, <ifcMode>]

Response(s)

^SCFG: "Serial/Ifc", <ifcMode>
OK
ERROR
+CME ERROR: <err>

Write Command

USB Device Descriptor:

AT^SCFG="Serial/USB/DDD" [, <deviceDescr>, [<descrIndex>], [<langId>], <vendorId>, <productId>, [<manufacturer>], [<product>], [<serialNo>]]

Response(s)

^SCFG: "Serial/USB/DDD" , <deviceDescr>, <descrIndex>, <langId>, <vendorId>, <productId>, <manufacturer>, <product>, <serialNo>
OK
ERROR
+CME ERROR: <err>

Write Command

Configuration of URC "+CIEV: call" Call Status Indication

AT^SCFG="URC/CallStatus/CIEV"[, <succ>]

Response(s)

^SCFG: "URC/CallStatus/CIEV", <succ>
OK
ERROR
+CME ERROR: <err>

Write Command

Configuration of URC "^SLCC" Call Status Indication

AT^SCFG="URC/CallStatus/SLCC"[, <sucs>]

Response(s)

^SCFG: "URC/CallStatus/SLCC", <sucs>
OK
ERROR
+CME ERROR: <err>

Write Command

URC indication in datamode via Ring line:

AT^SCFG="URC/Datamode/Ringline"[, <udri>]

Response(s)

^SCFG: "URC/Datamode/Ringline", <udri>
OK
ERROR
+CME ERROR: <err>

Write Command

URC indication via Ring line:

AT^SCFG="URC/Ringline"[, <uri>]

Response(s)

^SCFG: "URC/Ringline", <uri>
OK

Write Command (Continued)

URC indication via Ring line:

AT^SCFG="URC/Ringline"[, <uri>]

Response(s)

ERROR
+CME ERROR: <err>

Write Command

Duration of active RING line for URC indications:

AT^SCFG="URC/Ringline/ActiveTime"[, <urat>]

Response(s)

^SCFG: "URC/Ringline/ActiveTime", <urat>
OK
ERROR
+CME ERROR: <err>

Write Command

Userware autostart status:

AT^SCFG="Userware/Autostart"[, <upwd>, <ua>]

Response(s)

^SCFG: "Userware/Autostart"[, <ua>]
OK
ERROR
+CME ERROR: <err>

Write Command

Userware autostart application:

AT^SCFG="Userware/Autostart/AppName"[, <upwd>, <uaa>]

Response(s)

^SCFG: "Userware/Autostart/AppName"[, <uaa>]
OK
ERROR
+CME ERROR: <err>

Write Command

Userware autostart delay:

AT^SCFG="Userware/Autostart/Delay"[, <upwd>, <uad>]

Response(s)

^SCFG: "Userware/Autostart/Delay"[, <uad>]
OK
ERROR
+CME ERROR: <err>

Write Command

Userware configuration password:

AT^SCFG="Userware/Passwd"[, $<upwd>_{old}$, $<upwd>_{new}$, $<upwd>_{new}$]

Response(s)

^SCFG: "Userware/Passwd"
OK
ERROR
+CME ERROR: <err>

Write Command

Userware debug interface:

AT^SCFG="Userware/DebugInterface"[, <udbgif1>, <udbgif2>[, <udbgif3>]]

Response(s)

^SCFG: "Userware/DebugInterface", <udbgif1>, <udbgif2>, <udbgif3>
OK
ERROR
+CME ERROR: <err>

Write Command

Userware mode:

AT^SCFG="Userware/Mode"[, <umode>, <uurl>, <uport>]

Response(s)

^SCFG: "Userware/Mode", <umode>, <uurl>, <uport>
OK
ERROR
+CME ERROR: <err>

Write Command

Standard output of userware:

AT^SCFG="Userware/Stdout"[, <if>[, <intvalue>][, <filename>][, <logmode>]]

Response(s)

^SCFG: "Stdout" , <if>[, <intvalue>[, <filename>[, <logmode>]]]
OK
ERROR
+CME ERROR: <err>

Write Command

Userware mode:

AT^SCFG="Userware/Mode"[, <umode>, <uurl>, <uport>]

Response(s)

^SCFG: "Userware/Mode", <umode>, <uurl>, <uport>
OK
ERROR
+CME ERROR: <err>

Write Command

Standard output of userware:

AT^SCFG="Userware/Stdout"[, <if>[, <intvalue>][, <filename>][, <logmode>]]

Response(s)

^SCFG: "Stdout" , <if>[, <intvalue>[, <filename>[, <logmode>]]]
OK
ERROR
+CME ERROR: <err>

Write Command

Watchdog configuration and control:

AT^SCFG="Userware/Watchdog"[, <wd>]

Response(s)

^SCFG: "Watchdog", <wd>
OK
ERROR
+CME ERROR: <err>

3.3 AT^SIND Extended Indicator Control

Designed for extended event indicator control AT^SIND

- offers greater flexibility than the standard command AT+CIND,
- offers several extra indicators,
- can show the current status of all indicators supported by AT+CIND and AT^SIND,
- can be used to register or deregister the indicators of both commands,
- displays all indicator event reports via "+CIEV" URCs.

Presentation mode of the generated URCs is controlled via AT+CMER.

AT^SIND read command provides a list of all indicators supported by AT+CIND and AT^SIND. Each indicator is represented with its registration mode and current value.
AT^SIND write command can be used to select a single indicator in order to modify its registration and to view the current value.

Syntax

Test Command

AT^SIND=?

Response(s)

^SIND: (<indDescr>, list of supported <indValue>s)[, (<indDescr>, list of supported <indValue>s)[, ...]], (list of supported <mode>s)
OK

Read Command

AT^SIND?

Response(s)

^SIND: <indDescr>, <mode>[, <indValue>]
[^SIND: <indDescr>, <mode>[, <indValue>]]
...

In case of <indDescr>="eons"
^SIND: "eons", <mode>, <eonsOperator>, <servProvider>

In case of <indDescr>="nitz"
^SIND: "nitz", <mode>, <nitzUT>, <nitzTZ>, <nitzDST>

In case of <indDescr>="lsta"
^SIND: "lsta", <mode>, <lstaLevel>

In case of <indDescr>="is_cert"
^SIND: "is_cert", <mode>

OK
ERROR
+CME ERROR: <err>

Write Command

AT^SIND=<indDescr>, <mode>

Response(s)

^SIND: <indDescr>, <mode>[, <indValue>]

In case of: <indDescr>="eons" and <mode>=2
^SIND: "eons", <mode>, <indValue>, <eonsOperator>, <servProvider>

Write Command (Continued)

AT^SIND=<indDescr>, <mode>

Response(s)

In case of: <indDescr>="nitz" and <mode>=2

^SIND: "nitz", <mode>, <nitzUT>, <nitzTZ>, <nitzDST>

In case of: <indDescr>="is_cert" and <mode>=2

^SIND: "is_cert", <mode>

OK

ERROR

+CME ERROR: <err>

Write Command

AT^SIND="lsta", <mode>[, <lstaLevel>]

Response(s)

^SIND: "lsta", <mode>[, <lstaLevel>]

OK

ERROR

+CME ERROR: <err>

PIN	ASC0	ASC1	USB	MUX1	MUX2	MUX3	Charge	✈	Last
-	+	+	+	+	+	+	-	+	-

4.1 ATE AT Command Echo

ATE controls if the TC65i echoes characters received from TE during AT command state.

Syntax

Exec Command

ATE[<value>]

Response(s)

OK

PIN	ASC0	ASC1	USB	MUX1	MUX2	MUX3	Charge	✈	Last
-	+	+	+	+	+	+	-	+	-

Reference(s)

V.250

Parameter Description

<value>(num)(&W)(&V)

[0]	Echo mode off
1(&F)	Echo mode on

4.9 AT+IPR Bit Rate

AT+IPR allows to query and set the bit rate of the TC65i's asynchronous serial interfaces (UART).
The test command returns the values of supported automatically detectable bit rates and the values of the supported fixed bit rates.
The read command returns the the currently set <rate> value.
The write command specifies the bit rate to be used for the interface. Delivery bit rate value (<rate>) is 115200bps on ASC0 and 115200bps on ASC1. This setting will not be restored with AT&F.

If using a fixed bit rate, make sure that both TC65i and TE are configured to the same rate. A selected fixed bit rate takes effect after the write command returns OK and is stored in non-volatile memory. It is not recommended to set bit rates lower than 9600 bps in order to avoid timing problems (see Section 1.6, Communication between Customer Application and TC65i for details about timing).
In case of Autobaud mode (AT+IPR=0) the detected TE bit rate will not be saved and, therefore, needs to be resynchronized after any restart of the ME (for details refer Section 4.9.1, Autobauding). If Autobaud mode is activated, the ME will automatically recognize bit rate, character framing and parity format (refer AT+ICF) currently used by the TE.
In Multiplex mode the write command will not change the bit rate currently used, but the new bit rate will be stored and becomes active, when the ME is restarted.
If Java is running, the firmware will ignore any settings made with AT+IPR. Responses to the read, write or test command will be invalid or deliver "ERROR". Also refer "TC65i Java User's Guide" [3], Section "Configuring serial interface".
The current setting of AT+IPR will be preserved when you download firmware (i.e. a firmware update does not restore the factory setting) or in the event of power failure.

Syntax

Test Command

```
AT+IPR=?
```

Response(s)

```
+IPR:  (list of supported auto-detectable <rate>s) , (list of supported fixed-only <rate>s)
OK
```

Read Command

```
AT+IPR?
```

Response(s)

```
+IPR: <rate>
OK
```

Write Command

```
AT+IPR=<rate>
```

Response(s)

```
OK
ERROR
+CME ERROR: <err>
```

PIN	ASC0	ASC1	USB	MUX1	MUX2	MUX3	Charge	✈	Last
-	+	±	-	±	±	±	-	+	-

Reference(s)

V.250

Parameter Description

<rate>(num)(&V)

bit rate per second (bps)

0	Activates Autobaud mode. Not supported on ASC1. Not usable with Java. See Section 4.9.1, Autobauding for further details.
300	
600	
1200	
2400	
4800	
9600	
14400	
19200	
28800	
38400	

57600
115200
230400
460800
921600

Note

- Generally, AT+IPR should be used as a standalone command as specified in Section 1.5.2, Concatenating AT Commands. If nevertheless combinations with other commands on the same command line cannot be avoided, there are several constraints to be considered:
  - Avoid combinations with the AT commands listed in Section 1.5.2, Concatenating AT Commands.
  - Keep in mind that there shall be a minimum pause between two AT commands as specified in Section 1.6, Communication between Customer Application and TC65i.
  - If local echo is active (ATE1) and AT+IPR=x is entered with other commands you may encounter the following problem: If switching to the new bit rate takes effect while a response is being transmitted, the last bytes may be sent with the new bit rate and thus, not properly transmitted. The following commands will be correctly sent at the new bit rate.

5.1 AT+CPIN PIN Authentication

The AT+CPIN write command can be used to enter one of the passwords listed below. The read command can be used to check whether or not the ME is waiting for a password, or which type of password is required.

This may be for example the SIM PIN1 to register to the GSM network, or the SIM PUK1 to replace a disabled SIM PIN1 with a new one, or the PH-SIM PIN if the client has taken precautions for preventing damage in the event of loss or theft etc. If requested by the ME AT+CPIN may also be used for the SIM PIN2 or SIM PUK2.

If no PIN1 request is pending (for example if PIN1 authentication has been done and the same PIN1 is entered again) TC65i responds "+CME ERROR: operation not allowed"; no further action is required.

Each time a password is entered with AT+CPIN the module starts reading data from the SIM. The duration of reading varies with the SIM card. This may cause a delay of several seconds before all commands which need access to SIM data are effective. See Section 23.1, Restricted access to SIM data after SIM PIN authentication for further detail.

Syntax

Test Command

```
AT+CPIN=?
```

Response(s)

```
OK
```

Read Command

```
AT+CPIN?
```

Response(s)

```
+CPIN: <code>
OK
ERROR
+CME ERROR: <err>
```

Write Command

```
AT+CPIN=<pin>[, <new pin>]
```

Response(s)

```
OK
ERROR
+CME ERROR: <err>
```

PIN	ASC0	ASC1	USB	MUX1	MUX2	MUX3	Charge	✈	Last
-	+	+	+	+	+	+	-	+	-

Reference(s)

3GPP TS 27.007 [42]

Parameter Description

<pin>(str)

Password (string type), usually SIM PIN1.
If the requested password was a PUK, such as SIM PUK1 or PH-FSIM PUK or another password, then <pin> must be followed by <new pin>.

<new pin>(str)

If the requested code was a PUK: specify a new password or restore the former disabled password. See Section 5.1.1, What to do if PIN or password authentication fails? for more information about when you may need to enter the PUK.

<code>(text)

SIM PIN authentication	
READY	PIN has already been entered. No further entry needed.
SIM PIN	ME is waiting for SIM PIN1.
SIM PUK	ME is waiting for SIM PUK1 if PIN1 was disabled after three failed attempts to enter PIN1.
SIM PIN2	ME is waiting for PIN2. This is only applicable when an attempt to access a PIN2 related feature was acknowledged with +CME ERROR: 17 ("SIM PIN2 required"), for example when the client attempts to edit the FD phonebook). In this case the read command AT+CPIN? also prompts for SIM PIN2. Normally, the AT+CPIN2 command is intended for SIM PIN2.
SIM PUK2	ME is waiting for PUK2 to unblock a disabled PIN2. As above, this is only necessary when the preceding command was acknowledged with +CME ERROR: 18 ("SIM PUK2 required") and only if the read command AT+CPIN? also prompts for SIM PUK2. Normally, the AT+CPIN2 command is intended for SIM PUK2.
Phone security locks set by client or factory	
PH-SIM PIN	ME is waiting for phone-to-SIM card password if "PS" lock is active and the client inserts other SIM card than the one used for the lock. ("PS" lock is also referred to as phone or antitheft lock).
PH-SIM PUK	ME is waiting for Master Phone Code, if the above "PS" lock password was incorrectly entered three times.
PH-FSIM PIN	ME is waiting for phone-to-very-first-SIM card. Necessary when "PF" lock was set. When powered up the first time, ME locks itself to the first SIM card put into the card holder. As a result, operation of the mobile is restricted to this one SIM card (unless the PH-FSIM PUK is used as described below).
PH-FSIM PUK	ME is waiting for phone-to-very-first-SIM card unblocking password to be given. Necessary when "PF" lock is active and other than first SIM card is inserted.
PH-NET PUK	ME is waiting for network personalisation unblocking password
PH-NS PIN	ME is waiting for network subset personalisation password
PH-NS PUK	ME is waiting for network subset unblocking password
PH-SP PIN	ME is waiting for service provider personalisation password
PH-SP PUK	ME is waiting for service provider personalisation unblocking password
PH-C PIN	ME is waiting for corporate personalisation password
PH-C PUK	ME is waiting for corprorate personalisation un-blocking password

Notes

- Successful PIN authentication only confirms that the entered PIN was recognized and correct. The output of the result code OK does not necessarily imply that the mobile is registered to the desired network.
  Typical example: PIN was entered and accepted with OK, but the ME fails to register to the network. This may be due to missing network coverage, denied network access with currently used SIM card, no valid roaming agreement between home network and currently available operators etc.
  TC65i offers various options to verify the present status of network registration: For example, the AT+COPS command indicates the currently used network. With AT+CREG you can also check the current status and activate an unsolicited result code which appears whenever the status of the network registration changes (e.g. when the ME is powered up, or when the network cell changes).
- <pin> and <new pin> can also be entered in quotation marks (e.g. "1234").
- To check the number of remaining attempts to enter the passwords use the AT^SPIC command.
- See AT+CPWD and AT^SPWD for information on passwords.
- See AT+CLCK and AT^SLCK for information on lock types.

5.1.1 What to do if PIN or password authentication fails?

PIN1 / PUK1:
After three failures to enter PIN 1, the SIM card is blocked (except for emergency calls). +CME ERROR: 12 will prompt the client to unblock the SIM card by entering the associated PUK (= PIN Unblocking Key / Personal Unblocking Key). After ten failed attempts to enter the PUK, the SIM card will be invalidated and no longer operable (the device will respond with: +CME ERROR: 770, which stands for: SIM invalid - network reject). In such a case, the card needs to be replaced. PIN1 consists of 4 to 8 digits, PUK1 is an 8-digit code only.
To unblock a disabled PIN1 you have two options:

- You can enter AT+CPIN=PUK1,new PIN1.
- You can use the ATD command followed by the GSM code **05*PUK*newPIN*newPIN#;.

PIN2 / PUK2:
PIN2 prevents unauthorized access to the features listed in AT+CPIN2. The handling of PIN2 varies with the provider. PIN2 may either be a specific code supplied along with an associated PUK2, or a default code such as 0000. In either case, the client is advised to replace it with an individual code. Incorrect input of PUK2 will permanently block the additional features subject to PIN2 authentification, but usually has no effect on PIN1. PIN2 consists of 4 digits, PUK2 is an 8-digit code only.
To unblock a disabled PIN2 you have two options:

- You can enter AT+CPIN2=PUK2,new PIN2.
- You can use the ATD command followed by the GSM code **052*PUK2*newPIN2*newPIN2#;.

Phone lock:
If the mobile was locked to a specific SIM card (= "PS" lock or phone lock), the PUK that came with the SIM card cannot be used to remove the lock. After three failed attempts to enter the correct password, ME returns +CPIN: PH-SIM PUK (= response to read command AT+CPIN?), i.e. it is now waiting for the Master Phone Code. This is an 8-digit device code associated to the IMEI number of the mobile which can only by obtained from the manufacturer or provider. When needed, contact Cinterion Wireless Modules GmbH and request the Master Phone Code of the specific module.
There are two ways to enter the Master Phone code:

- You can enter AT+CPIN=Master Phone Code
- You can use the ATD command followed by the GSM code *#0003*Master Phone Code#;.

Usually, the Master Phone Code will be supplied by mail or e-mail. If the received number is enclosed in the *# codes typically used for the ATD option, it is important to crop the preceding *#0003* characters and the appended #.
Example: You may be given the string *#0003*12345678#. When prompted for the PH-SIM PUK simply enter 12345678.
If incorrectly input, the Master Phone Code is governed by a specific timing algorithm: (n-1)*256 seconds (see table below). The timing should be considered by system integrators when designing an individual MMI.

Number of failed attempts	Time to wait before next input is allowed
1st failed attempt	No time to wait
2nd failed attempt	4 seconds
3rd failed attempt	3 * 256 seconds
4th failed attempt	4 * 256 seconds
5th failed attempt	5 * 256 seconds
6th failed attempt and so forth	6 * 256 seconds and so forth

SIM locks:
These are factory set locks, such as "PF", "PN", "PU", "PP", "PC". An 8-digit unlocking code is required to operate the mobile with a different SIM card, or to lift the lock. The code can only be obtained from the provider.
Failure to enter the password is subject to the same timing algorithm as the Master Phone Code (see Table above).

Call barring:
Supported modes are "AO", "OI", "OX", "AI", "IR", "AB", "AG", "AC". If the call barring password is entered incorrectly three times, the client will need to contact the service provider to obtain a new one.

Related sections:
"+CME ERROR: <err>" values are specified at Section 2.12.1, CME/CMS Error Code Overview. For further instructions and examples see AT+CLCK, AT^SLCK, AT+CPWD and AT^SPWD.
For a complete list of Star-Hash codes please refer Section 23.2, Star-Hash (*#) Network Commands.

8.1 AT+COPN Read operator names

The AT+COPN command returns the list of operator names from the ME. Each operator code <numericn> that has an alphanumeric equivalent <alphan> in the ME memory is returned. See also: AT^SPLM.

Syntax

Test Command

```
AT+COPN=?
```

Response(s)

```
OK
ERROR
+CME ERROR: <err>
```

Exec Command

```
AT+COPN
```

Response(s)

```
+COPN: <numericn>,<alphan>
[+COPN: ...]
OK
ERROR
+CME ERROR: <err>
```

PIN	ASC0	ASC1	USB	MUX1	MUX2	MUX3	Charge	→	Last
+	+	+	+	+	+	+	-	+	-

Reference(s)

3GPP TS 27.007 [42]

Parameter Description

<numericn>(str)

Operator in numeric format; GSM location area identification number.

<alphan>(str)

Operator in long alphanumeric format; can contain up to 26 characters.

8.2 AT+COPS Operator Selection

AT+COPS queries the present status of the TC65i's network registration and allows to determine whether automatic or manual network selection shall be used. Additional service is available with AT^SOPS.

Three operator selection modes are available:

- Automatic
  TC65i searches for the home operator automatically. If successful the TC65i registers to the home network. If the home network is not found, TC65i goes on searching. If a permitted operator is found, TC65i registers to this operator.
  If no operator is found the TC65i remains unregistered.
- Manual
  Desired operator can be determined using the AT+COPS write command. If the operator is found, TC65i registers to it immediately. If the selected operator is forbidden, the TC65i remains unregistered.
- Manual/automatic
  The ME first tries to find the operator determined via AT+COPS write command. If the ME is able to register to this operator, it enters the manual operator selection mode. If it cannot find this operator or fails to register to this operator, then it enters the automatic operator selection mode and starts to select the home operators network or another (permitted) one. If the ME is registered and the manually selected network is not available, the ME will remain registered without further result code notification.

The most recently entered operator selection mode is still valid after the ME was restarted (power-off/on).

The AT+COPS test command conists of several parameter sets, each representing an operator present in the network.
Each set contains the following information:

- an integer indicating the availability of the operator,
- long alphanumeric format of the operator's name and
- numeric format representation of the operator.

Any of the parameters may be unavailable and will then be an empty field (,,). The list of operators comes in the following order: Home network, networks referenced in SIM and other networks.
The operator list is followed by a list of the supported <mode>s and <format>s. These lists are delimited from the operator list by two commas.
If the test command is used during an ongoing GPRS transfer, traffic will be interrupted for up to one minute.

The AT+COPS read command returns the current <mode> and the currently selected operator. If no operator is selected, <format> and <opName> are omitted.

The AT+COPS write command forces an attempt to select and register to the GSM network operator (see note below). If the selected operator is not available, no other operator will be selected (except <mode>=4). The selected operator name <format> will apply to further read commands, too.

Command settings are effective over all serial interfaces of the TC65i.

Syntax

Test Command

AT+COPS=?

Response(s)

+COPS: [list of supported (<opStatus>, long alphanumeric <opName>,,numeric <opName>)s], , (list of supported <mode>s), (list of supported <format>s)
OK
ERROR
+CME ERROR: <err>

Read Command

AT+COPS?

Response(s)

+COPS:<mode>[, <format>[, <opName>]]
OK
ERROR
+CME ERROR: <err>

Write Command

AT+COPS=<mode>[, <format>[, <opName>]]

Response(s)

OK
ERROR
+CME ERROR: <err>

PIN	ASC0	ASC1	USB	MUX1	MUX2	MUX3	Charge	✈	Last
±	+	+	+	+	+	+	-	-	-

Reference(s)

3GPP TS 27.007 [42]

Parameter Description

<opStatus>(num)

Operator Status

0	Unknown
1	Operator available
2	Current operator
3	Operator forbidden

<opName>(str)(&V)

Operator Name

If test command: Operator name in long alphanumeric format and numeric format.
If read command: Operator name as per <format>.
If write command: Operator name in numeric format.

<mode>(num)(&V)

Parameter values 0 and 1 are stored non-volatile in the TC65i.

0(D)	Automatic mode; <opName> field is ignored.
1	Manual operator selection Write command requires <opName> in numeric format, i.e. <format> shall be 2. Read command returns the current <mode> and the currently selected <opName>. If no operator is selected, <format> and <opName> are omitted.
2	Manually deregister from network and remain unregistered until <mode>=0 or 1 or 4 is selected.
3	Set only <format> (for AT+COPS read command).
4	Automatic / manual selection; if manual selection fails, automatic mode (<mode>=0) is entered (<opName> field will be present).

<format>(num)(&W)(&V)

Parameter is global for all instances.

0(&F)	Long alphanumeric format of <opName>. Can be up to 26 characters long.
2	Numeric format of <opName>. This is the GSM Location Area Identification (LAI) number, which consists of the 3-digit Mobile Country Code (MCC) plus the 2- or 3-digit Mobile Network Code (MNC).

Notes

- It is not recommended to use the AT+COPS command before passing the CHV (card holder verification) / SIM PIN1 verification. In case of entering of AT+COPS= 0 before PIN1 verification the ME will answer with OK and does not try to register to the network. Also, the test command should only be used after PIN1 authentication.
- It's possible to apply a 5- or 6-digit LAI for numeric format of <opName> parameter. Please use the correct 2- or 3-digit Mobile Network Code. Otherwise an unintended PLMN could be selected.
- It is not recommended to use the AT+COPS write and test command while TC65i is searching for a new operator. In this case the ME will answer with ERROR. Please use AT+CREG to verify the network registration status.

8.3 AT^SOPS Extended Operator Selection

AT^SOPS queries the present status of the TC65i's network registration. Since basic operator selection services are available with AT+COPS this command uses the methods of the Enhanced Operator Name String (EONS) specification while handling operator name strings. Additional EONS related information is available with AT^SIND.

AT^SOPS test command lists sets of five parameters, each representing an operator present in the network.
A set consists of

1. an integer indicating the availability of the operator,
2. specification of the source of the operator name <eonsOperator>,
3. operator name according to EONS Table,
4. Service Provider Name from the SIM Service Table and
5. numeric format representation of the operator.

Any of the parameters may be unavailable and will then be an empty field (,,).
The list of operators comes in the following order: Home network, networks referenced in SIM and other networks.
If the test command is used during an ongoing GPRS transfer, traffic will be interrupted for up to a minute.

Syntax

Test Command

AT^SOPS=?

Response(s)

^SOPS: [list of present operator(<opStatus>, <eonsType>, <eonsOperator>, <servProvider>, <opName>)s], , (), ()
OK
ERROR
+CME ERROR: <err>

PIN	ASC0	ASC1	USB	MUX1	MUX2	MUX3	Charge	✈	Last
+	+	+	+	+	+	+	-	-	-

Parameter Description

<opStatus>(num)	
Status	
0	unknown
1	operator available
2	current operator
3	operator forbidden

<eonsType>(num)

Specification of the source of the operator name <eonsOperator>. Details of EONS-supplied operator name types are available at AT^SIND.

<eonsOperator>(+CSCS)

Operator name; format depends on the source of the operator name, specified by <eonsType>. Can be up to 24 characters long.

<servProvider>(str)(+CSCS)

Service Provider Name according to setting of Service No. 17 in the SIM Service Table (EF_{SST}). Can be up to 16 characters long.

<opName>

Operator

Operator name in numerical presentation contains the GSM Location Area Identification (LAI) number, which consists of the 3-digit Mobile Country Code (MCC) plus the 2- or 3-digit Mobile Network Code (MNC).

Note

- It is not recommended to use the AT^SOPS test command while TC65i is searching for a new operator. In this case the module will answer with ERROR. Please use AT+CREG to verify the network registration status.

8.4 AT+CREG Network Registration Status

AT+CREG serves to monitor the TC65i's network registration status. For this purpose the read command or URC presentation mode are available.

Syntax

Test Command

AT+CREG=?

Response(s)

+CREG: (list of supported<urcMode>s)
OK

Read Command

AT+CREG?

Response(s)

+CREG: <urcMode>, <regStatus>[, <netLac>, <netCellId>]
OK
ERROR
+CME ERROR: <err>

Write Command

AT+CREG=[<urcMode>]

Response(s)

OK
ERROR
+CME ERROR: <err>

PIN	ASC0	ASC1	USB	MUX1	MUX2	MUX3	Charge	✈	Last
-	+	+	+	+	+	+	-	-	-

Reference(s)

3GPP TS 27.007 [42]

Parameter Description

<urcMode>(num)(&W)(&V)	
[0](&F)	Disable +CREG URC.
1	Enable URC +CREG:<regStatus> to report status of network registration.
2	Enable URC +CREG:<regStatus>[,<netLac>,<netCellId>] to report status of network registration including location information. Optional parameters <netLac> and <netCellId> will not be updated during calls.

<regStatus>(num)(&V)	
0	Not registered, ME is currently not searching for new operator Normally, status 0 occurs temporarily between two network search phases (status 2). However, if it persists, one the following reasons may apply: • Automatic network selection is active, but probably there is - no SIM card available - no PIN entered - no valid Home PLMN entry found on the SIM • Manual network selection is active and the selected network is available, but login fails due to one of the following reasons: - #11 ... PLMN not allowed - #12 ... Location area not allowed - #13 ... Roaming not allowed in this location area In either case, user intervention is required. Yet, emergency calls can be made if any network is available.
1	Registered to home network
2	Not registered, but ME is currently searching for a new operator ME searches for an available network. Failure to log in until after more than a minute may be due to one of the following reasons: • No network available or insufficient Rx level. • ME has no access rights to the networks available. • Networks from the SIM list of allowed networks are around, but login fails due to one of the following reasons: - #11 ... PLMN not allowed - #12 ... Location area not allowed - #13 ... Roaming not allowed in this location area After this, the search will be resumed (if automatic network search is enabled). • The Home PLMN or an allowed PLMN is available, but login is rejected by the cell (reasons: Access Class or LAC). If at least one network is available, emergency calls can be made.
3	Registration denied • Authentication or registration fails after Location Update Reject due to one of the following reasons: - #2 ... IMSI unknown at HLR - #3 ... Illegal MS - #6 ... Illegal ME Either the SIM or the ME are unable to log into any network. No further attempt is made to search or log into a network. User intervention is required. Emergency calls can be made, if any network is available.
4	Unknown (not used)
5	Registered, roaming ME is registered at a foreign network (national or international network)

<netLac>(str)
Two byte location area code in hexadecimal format (e.g. "00C3" equals 195 in decimal).

<netCellId>(str)

Two byte cell ID in hexadecimal format.

Note

- After the "+CREG: 1" (or "+CREG: 5") URC and before the SIM notification URC "^SSIM READY" and/or "+CIEV: simstatus" it is not sure that outgoing and incoming calls can be made and short message functions executed. Emergency calls are possible.
  Outgoing and incoming calls are always possible AFTER having received the "+CREG: 1" (or "+CREG: 5") and SIM notification URC "^SSIM READY" and/or "+CIEV: simstatus".
  See also Section 23.1, Restricted access to SIM data after SIM PIN authentication.

Example

```
AT+CREG=2                       Activate extended URC mode.
OK
AT+COPS=0                       Force ME to automatically search a network operator.
OK
+CREG: 2                        URC reports that ME is currently searching.
+CREG: 1,"0145","291A"          URC reports that operator has been found.
```

8.5 AT+CSQ Signal quality

The AT+CSQ execute command indicates the received signal strength <rssi> and the channel bit error rate <ber>.

Syntax

Test Command

```
AT+CSQ=?
```

Response(s)

```
+CSQ: (list of supported<rssi>s), (list of supported<ber>s)
OK
```

Exec Command

```
AT+CSQ
```

Response(s)

```
+CSQ: <rssi>,<ber>
OK
```

PIN	ASC0	ASC1	USB	MUX1	MUX2	MUX3	Charge	✈	Last
-	+	+	+	+	+	+	-	-	-

Reference(s)

3GPP TS 27.007 [42]

Parameter Description

<rssi>(num)	
0	-113 dBm or less
1	-111 dBm
2..30	-109... -53 dBm
31	-51 dBm or greater
99	not known or not detectable

<ber>(num)	
0..7	as RXQUAL values in the table in 3GPP TS 45.008 [46] section 8.2.4.
99	not known or not detectable

Note

- After using network related commands such as AT+CCWA, AT+CCFC, AT+CLCK, users are advised to wait 3s before entering AT+CSQ. This is recommended to be sure that any network access required for the preceding command has finished.

19.1 AT+CCLK Real Time Clock

Syntax

Test Command

AT+CCLK=?

Response(s)

OK

Read Command

AT+CCLK?

Response(s)

+CCLK: <time>
OK

Write Command

AT+CCLK=<time>

Response(s)

OK
ERROR
+CME ERROR: <err>

PIN	ASC0	ASC1	USB	MUX1	MUX2	MUX3	Charge	✈	Last
-	+	+	+	+	+	+	+	+	-

Reference(s)

3GPP TS 27.007 [42]

Parameter Description

<time>(str)

Format is "yy/mm/dd,hh:mm:ss", where the characters indicate the two last digits of the year, followed by month, day, hour, minutes, seconds; for example 6th of July 2005, 22:10:00 hours equals to "05/07/06,22:10:00" Factory default is "02/01/01,00:00:00"

Notes

- <time> is retained if the device enters the Power Down mode via AT^SMSO.
- <time> will be reset to its factory default if power is totally disconnected. In this case, the clock starts with <time>= "02/01/01,00:00:00" upon next power-up.
- Each time TC65i is restarted it takes 2s to re-initialize the RTC and to update the current time. Therefore, it is recommended to wait 2s before using the commands AT+CCLK and AT+CALA (for example 2s after ^SYS-START has been output).

11.3 AT+CGATT GPRS attach or detach

Syntax

Test Command

AT+CGATT=?

Response(s)

+CGATT: (list of supported <state>s)
OK

Read Command

AT+CGATT?

Response(s)

+CGATT: <state>
OK

Write Command

AT+CGATT=[<state>]

Response(s)

OK
ERROR
+CME ERROR: <err>

PIN	ASC0	ASC1	USB	MUX1	MUX2	MUX3	Charge	✈	Last
+	+	+	+	+	+	+	-	-	-

Reference(s)

3GPP TS 27.007 [42]

Parameter Description

<state>(num)	
Indicates the state of GPRS attachment.	
0(P)	detached
[1]	attached

Notes

- If the MT is in dedicated mode, write command returns "+CME ERROR: operation temporary not allowed".
- When the module is GPRS attached and a PLMN reselection occurs to a non-GPRS network or to a network where the SIM is not subscribed to for using GPRS, the resulting GMM (GPRS mobility management) state according to GSM 24.008 is REGISTERED/NO CELL, meaning that the read command will still show <state>=1.

11.12 AT+CGREG GPRS Network Registration Status

AT+CGREG write command enables presentation of URC "+CGREG: <stat>" when <n>=1 and ME's GPRS network registration status changes, or URC "+CGREG: <stat>, <lac>, <ci>" when <n>=2 and the current network cell changes.
AT+CGREG read command queries the current URC presentation status and <stat> which shows whether the network has currently indicated the registration of the ME. Location information elements <lac> and <ci> are returned only if <n>=2 and ME is registered to the network.

Syntax

Test Command

```
AT+CGREG=?
```

Response(s)

```
+CGREG: (list of supported <n>s)
OK
```

Read Command

```
AT+CGREG?
```

Response(s)

```
+CGREG: <n>, <stat>[, <lac>, <ci>]
OK
```

Write Command

```
AT+CGREG=[<n>]
```

Response(s)

```
OK
ERROR
+CME ERROR: <err>
```

PIN	ASC0	ASC1	USB	MUX1	MUX2	MUX3	Charge	✈	Last
+	+	+	+	+	+	+	-	+	-

Reference(s): 3GPP TS 27.007 [42]

Parameter Description

<n>(num)	
0(&F)(P)	Disable network registration unsolicited result code
1	Enable network registration URC "+CGREG: <stat>"
2	Enable network registration URC "+CGREG: <stat>, <lac>, <ci>"

<stat>(num)	
0	Not registered, ME is not currently searching an operator to register to. The ME is in GMM state GMM-NULL or GMM-DEREGISTERED-INITIATED. GPRS service is disabled, the ME is allowed to attach to GPRS if requested by the user.
1	Registered, home network. The ME is in GMM state GMM-REGISTERED or GMM-ROUTING-AREA-UPDATING-INITIATED INITIATED on the home PLMN
2	Not registered, but ME is currently trying to attach or searching an operator to register to. The ME is in GMM state GMM-DEREGISTERED or GMM-REGISTERED-INITIATED. The GPRS service is enabled, but an allowable PLMN is currently not available. The ME will start a GPRS attach as soon as an allowable PLMN is available.
3	Registration denied. The ME is in GMM state GMM-NULL. The GPRS service is disabled, the ME is not allowed to attach to GPRS if requested by the user.
4	Unknown
5	Registered, roaming. The ME is in GMM state GMM-REGISTERED or GMM-ROUTING-AREA-UPDATING-INITIATED on a visited PLMN.

<lac>(str)

Two byte location area code in hexadecimal format.

<ci>(str)

Two byte cell ID in hexadecimal format.

Note

- When the module is GPRS attached and a PLMN reselection occurs to a non-GPRS network or to a network where the SIM is not subscribed to for using GPRS, the resulting GMM (GPRS mobility management) state according to GSM 24.008 is REGISTERED/NO CELL, meaning that the read command will still show <stat>=1 or <stat>=5.

11.22 Using GPRS AT commands (Examples)

Examples

EXAMPLE 1

Defining and using a Context Definition ID (CID):
Every time a CID is used as a parameter for a GPRS command the CID has to be defined before by using the AT+CGDCONT command. To get the parameter of a CID use the AT+CGDCONT read option. If the response of 'AT+CGDCONT?' is OK only, there is no CID defined.

```
AT+CGDCONT?
OK                                                    There is no CID defined
```

All parameters of the CID are initiated by NULL or not present values, and the CID itself is set to be undefined. To define a CID use the AT+CGDCONT command with at least one CID parameter. At the moment the mobile supports CID 1 and CID 2 by using the AT+CGDCONT command.

Define CID 1 and set the PDP type to IP, access point name and IP address are not set:

```
AT+CGDCONT=1,"IP"
OK
```

Define CID 2 and sets PDP type, APN and IP addr:

```
AT+CGDCONT=2,"IP", "internet.t-d1.gprs", 111.222.123.234
OK
```

A following read command will respond:

```
AT+CGDCONT?
+CGDCONT:1,"IP","","",0,0
+CGDCONT:2,"IP","internet.t-d1.gprs",111.222.123.234
OK
```

Set the CID 1 to be undefined:

```
AT+CGDCONT=1
OK
```

A following read command will respond:

```
AT+CGDCONT?
+CGDCONT:2,"IP","internet.t-d1.gprs",111.222.123.234
OK
```

EXAMPLE 2

Quality of Service (QoS) is a special parameter of a CID which consists of several parameters itself.
The QoS consists of
- the precedence class
- the delay class
- the reliability class
- the peak throughput class
- the mean throughput class
and is divided in "requested QoS" and "minimum acceptable QoS".

All parameters of the QoS are initiated by default to the "network subscribed value (= 0)" but the QoS itself is set to be undefined. To define a QoS use the AT+CGQREQ or AT+CGQMIN command.

Overwrite the precedence class of QoS of CID 1 and set the QoS of CID 1 to be present:

```
AT+CGQREQ=1,2
OK
```

A following read command will respond:

```
AT+CGQREQ?
+CGQREQ: 1,2,0,0,0,0
OK
```

All QoS values of CID 1 are set to network subscribed now, except precedence class which is set to 2. Now set the QoS of CID 1 to not present:

```
AT+CGQREQ=1
OK
```

Once defined, the CID it can be activated. To activate CID 2 use:

```
AT+CGACT=1,2
OK
```

If the CID is already active, the mobile responds OK at once.
If no CID and no STATE is given, all defined CIDs will be activated by:

If the CID is already active, the mobile responds OK at once.
If no CID and no STATE is given, all defined CIDs will be activated by:

```
AT+CGACT=
OK
```

If no CID is defined the mobile responds +CME ERROR: invalid index
Remark: If the mobile is NOT attached by AT+CGATT=1 before activating, the attach is automatically done by the AT+CGACT command.

After defining and activating a CID it may be used to get online by:

`AT+CGDATA="PPP",1` `CONNECT`	The mobile is connected using the parameters of CID 1.
`AT+CGDATA=` `CONNECT`	The mobile is connected using default parameters (<L2P>="PPP" and <cid> as described for command AT+CGDATA).

The mobile supports Layer 2 Protocol (L2P) PPP only.
Remark: If the mobile is NOT attached by AT+CGATT=1 and the CID is NOT activated before connecting, attaching and activating is automatically done by the AT+CGDATA command.

11.23 Using the GPRS dial command ATD

Example

In addition to the GPRS AT commands you can use the "D" command to dial into to the GPRS network.

There are two GPRS Service Codes for the ATD command: Values 98 and 99.
Examples:

ATD*99# CONNECT	Establish a connection by service code 99.
ATD*99*123.124.125.126*PPP*1# CONNECT	Establish a connection by service code 99, IP address 123 and L2P = PPP and using CID 1. The CID has to be defined by AT+CGDCONT.
ATD*99**PPP# CONNECT	Establish a connection by service code 99 and L2P = PPP.
ATD*99***1# CONNECT	Establish a connection by service code 99 and using CID 1.
ATD*99**PPP*1# CONNECT	Establish a connection by service code 99 and L2P = PPP and using CID 1. The CID has to be defined by AT+CGDCONT.
ATD*98# CONNECT	Establish a connection by service code 98.
ATD*98*1# CONNECT	Establish an IP connection by service code 98 using CID 1. The CID has to be defined by AT+CGDCONT.

21.1 AT^SJRA Run Java Application

The AT^SJRA write command launches the Java application.

Syntax

Test Command

AT^SJRA=?

Response(s)

("IMlet path")

Write Command

AT^SJRA=<appName>

Response(s)

^SJRA:
OK
If not successful:
ERROR
+CME ERROR: <err>

PIN	ASC0	ASC1	USB	MUX1	MUX2	MUX3	Charge	✈	Last
-	+	+	+	+	+	+	-	+	-

Parameter Description

<appName>(str)

Path of the Java application

The application name must be given as a fully qualified pathname (a:/.../...) to the jar/jad file containing the desired application.
The local flash file system is identified by: A:. Directory separator is "/" (002Fh).
Example: *A:/java/jam/example/helloworld/helloworld.jar*

Notes

- As an alternative, the Java application can be enabled to start up automatically whenever TC65i is getting started. Use the AT^SCFG command to make all the settings need for the Java autostart mode.
- When the Java application starts, all current calls will be terminated.

IV. Manuales de SDK Cinterion TC65i [70]

2 Overview

The ME features an ultra-low profile and low-power consumption for data (CSD and GPRS), voice, SMS and fax. Java technology and several peripheral interfaces on the module allow you to easily integrate your application.

This document explains how to work with the ME, the installation CD and the tools provided on the installation CD.

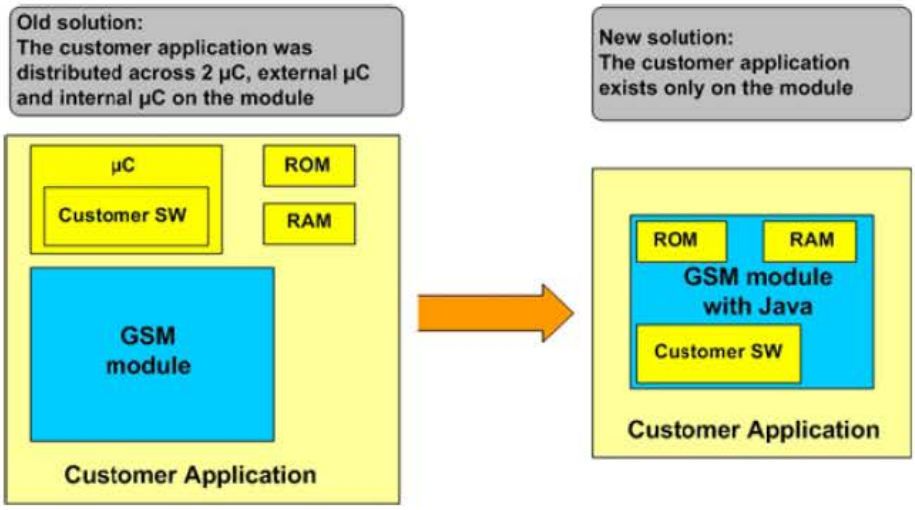

Figure 1: Overview

4.2 Interfaces

4.2.1 ASC0 - Serial Device

ASC0, an Asynchronous Serial Controller, is a 9-wire serial interface. It is described in [2]. Without a running Java application the module can be controlled by sending AT commands over ASC0. Furthermore, ASC0 is designed for transferring files from the development PC to the module. When a Java application is started, ASC0 can be used as an RS-232 port or/and System.out. Refer to [3] for details.

4.2.2 General Purpose I/O

There are ten I/O pins that can be configured for general purpose I/O. One pin can be configured as a pulse counter. All lines can be accessed under Java by AT commands or a Java API. See [1] and [2] for information about usage and startup behavior.

4.2.3 DAC/ADC

There are two analogue input lines and one analogue output line. They are accessed by AT commands or via a Java API. See [1] and [2] for details.

4.2.4 ASC1

ASC1 is the second serial interface on the module. This is a 4-pin interface (RX, TX, RTS, CTS). It can be used as a second AT interface when a Java application is not running or by a running Java application as RS-232 port or/and System.out.

4.2.7 JVM Interfaces

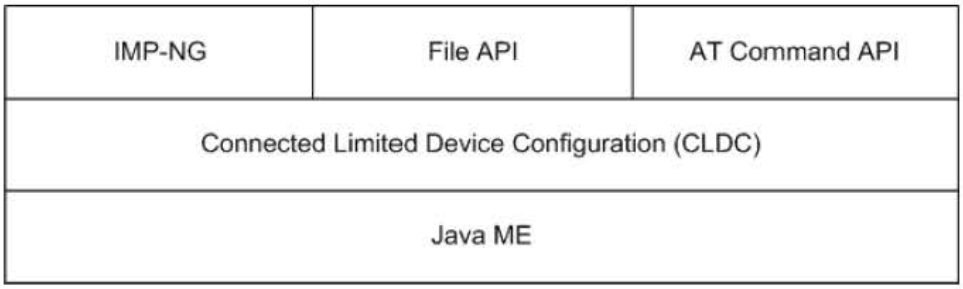

Figure 8: Interface Configuration

4.3 Data Flow of a Java Application Running on the Module

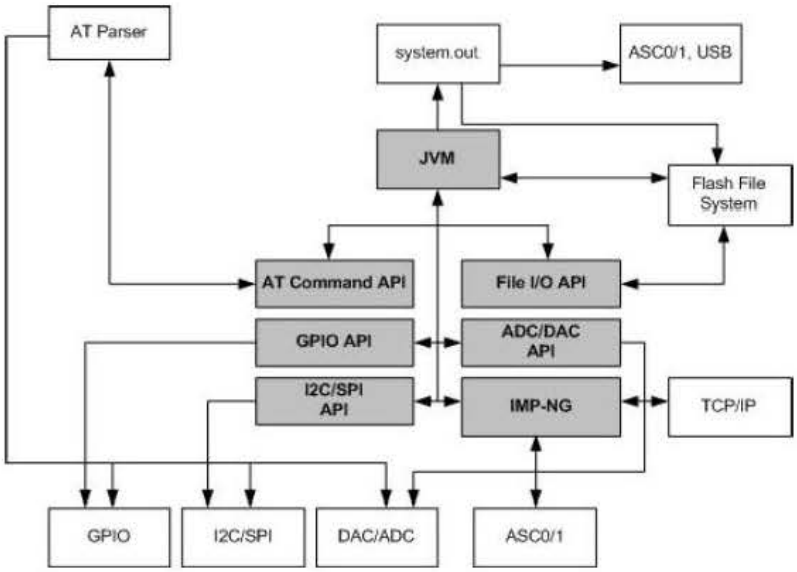

Figure 9: Data flow of a Java application running on the module.

The diagram shows the data flow of a Java application running on the module. The data flow of a Java application running in the debug environment can be found in Figure 24.
The compiled Java applications are stored as JAR files in the Flash File System of module. When the application is started, the JVM interprets the JAR file and calls the interfaces to the module environment.

5.7.1 Automatic Shutdown

The ME is switched off automatically in different situations:

- under- or overtemperature
- under- or overvoltage

Appropriate warning messages transmitted by the ME to the host application are implemented as URCs. To activate the URCs for temperature conditions use the AT^SCTM command. Undervoltage and overvoltage URCs are generated automatically when fault conditions occur.

For further detail refer to the commands AT^SCTM and AT^SBC described in the AT Command Set [1]. In addition, a description of the shutdown procedures can be found in [2].

5.7.2 Manual Shutdown

The module can be switched off manually with the AT command, AT^SMSO. In this case the midlets destroyApp method is called and the application has 5s time to clean up and call the notifydestroy method. After the 5s the VM is shut down.

5.7.3 Restart after Switch Off

When the module is switched off without setting an alarm time (see the AT Command Set [1]), e.g. after a power failure, external hardware must restart the module with the Ignition line (IGT). The Hardware Interface Description [2] explains how to handle a switched off situation.

5.12.3.1 Plain Serial Interface

Scenario: A device is connected to ASC0 (refer to Section 4.2.4). The Java application must handle data input and output streams. A simple Java application (with only one thread) which loops incoming data directly to output, reaches data rates up to 180kbit/s. Test conditions: hardware flow control enabled (<autorts> and <autocts>), 8N1, and baud rate on ASC0 set to 230kbaud (→ theoretical maximum: 184kbit/s net data rate).

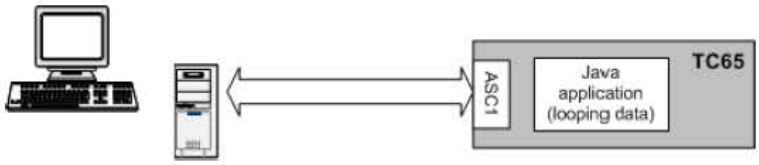

Figure 17: Scenario for testing data rates on ASC1

11.5.4 Sign a MIDlet

Use the tool "jadtool.jar" to sign a Java MIDlet. This program is in the folder "wkt\bin".

```
java -jar jadtool.jar -addjarsig -jarfile helloworld.jar
                      -inputjad helloworld.jad
                      -outputjad helloworld.jad
                      -alias keyname -storepass keystorepassword
                      -keypass keypassword -keystore customer.ks
                      -encoding UTF-8
```

11.6 Attention

The central element of Java Security is the **private key**. If Java Security is activated and you lose the private key, then the **module is destroyed**. You have no chance of deactivating Java Security, downloading of a new Midlet or starting any other operation concerning Java Security. To prevent problems you are strongly advised to **secure the private key**.

12.1.3 ATCommandListener Interface

The *ATCommandListener* interface implements callback functions for:
- URCs
- Changes of the serial interface signals RING, DCD and DSR
- Opening and closing of data connections

The user must create an implementation class for *ATCommandListener* to receive AT events. The *ATEvent* method of this class must contain the processing code for the different AT-Events (URCs). The *RINGChanged, DCDChanged, DSRChanged* and *CONNChanged* methods should contain the processing code for possible state changes. This code shall leave the listener context as soon as possible (i.e. running a new thread). While the callback method does not return it cannot be called again if changes occur.

The *CONNChanged* method indicates the start and the end of data connections. During a data connection it is possible to transfer data with the I/O stream methods (see Section 12.1.1.3). Some data services (i.e. FTP) transfer the data so quickly, that the *CONNChanged* start and close events are received even parallel to the response of the AT command which originated the data connection. The user's application must realize, that the data connection had taken place and read the data with the I/O stream read methods afterwards.

BIBLIOGRAFÍA Y REFERENCIAS

[1] DLMS, «Documentación del protocolo DLMS,» [En línea]. Available: http://www.dlms.com/documentation/overviewexcerptsofthedlmsuacolouredbooks/dlmsdocumentationarchive.html. [Último acceso: 14 Septiembre 2015].

[2] Modbus, «Documentación del protocolo Modbus,» [En línea]. Available: http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf. [Último acceso: 14 Septiembre 2015].

[3] MTXTunnel, «FAQ MTXTunnel,» [En línea]. Available: http://www.mtxtunnel.com/faq-mtxtunnel/. [Último acceso: 12 Septiembre 2015].

[4] MTXTunnel, «Manual de Usuario del MTX-Tunnel v8.10,» [En línea]. Available: http://www.mtxtunnel.com/Downloads/MTX-Tunnel-Manual-sp.pdf. [Último acceso: 12 Septiembre 2015].

[5] MTXTunnel, «Modems con version de MTXTunnel,» [En línea]. Available: http://mtxm2m.com//soporte/. [Último acceso: 12 Septiembre 2015].

[6] MTXTunnel, «Software detallado MTXTunnel,» [En línea]. Available: http://mtxm2m.com//productos/mtx-tunnel-software/. [Último acceso: 12 MTXTunnel 2015].

[7] Meters and More, «Página principal de Meters and More,» [En línea]. Available: http://www.metersandmore.com/. [Último acceso: 12 MTXTunnel 2015].

[8] ENDESA, «Información sobre los contadores de telegestión instalados por ENDESA,» [En línea]. Available: http://es.slideshare.net/CanalEndesa/los-contadores-inteligentes-de-endesa. [Último acceso: 12 Septiembre 2015].

[9] Meters and More, «Especificación del protocolo Meters and More,» [En línea]. Available: http://www.metersandmore.com/wp-content/uploads/2014/04/20140318-MM-Press-Release.pdf. [Último acceso: 12 Septiembre 2015].

[10] CENELEC, «FAQ de CENELEC,» [En línea]. Available: http://www.cenelec.eu/faq/faq_entry.htm. [Último acceso: 12 Septiembre 2015].

[11] CENELEC, «Global partners de CENELEC,» [En línea]. Available: http://www.cenelec.eu/aboutcenelec/whoweare/globalpartners/iec.html. [Último acceso: 12 Septiembre 2015].

[12] 4CTT, «Manual de usuario del concentrador 4CTT,» [En línea]. Available: http://www.meteringsolutions.ziv.es/documentacion/manuales/castellano/LCCT1005Av01.pdf. [Último acceso: 12 Septiembre 2015].

[13] S. Grids, «Congreso de Smart grids en Madrid, 2012,» [En línea]. Available: https://www.smartgridsinfo.es/images/SMARTGRIDSINFO/media/content/20150216-libro-comunicaciones-i-congreso-smart-grids.pdf. [Último acceso: 12 Septiembre 2015].

[14] «Listado de iniciativas referentes a Smart grids,» [En línea]. Available: http://stargrid.eu/downloads/2013/07/STARGRID_Industry_Initiatives_Documentation_Map_v1.0.pdf. [Último acceso: 12 Septiembre 2015].

[15] MTX, «Características MTX65i,» [En línea]. Available: http://www.matrix.es/Comunicaciones-M2M-inalambricas-RFID-Networking-Industrial/GSM-GPRS-3G/Terminales/Modems-GPRS-GSM-programables-en-Java/Modelos/MTX-65i. [Último acceso: 12 Septiembre 2015].

[16] gemalto, «Características del módulo TC65i,» [En línea]. Available: http://www.gemalto.com/m2m/solutions/modules-terminals/other-families/tc65i. [Último acceso: 12 Septiembre 2015].

[17] ETSI, [En línea]. Available: http://www.etsi.org/technologies-clusters/technologies/m2m. [Último acceso: 20 Septiembre 2015].

[18] Cirwatt, «Especificaciones del contador monofásico Cirwatt B200RCP,» [En línea]. Available: http://circutor.es/es/productos/metering/contadores-de-energia-electrica-multifuncion/contadores-monofasicos/serie-cirwatt-b200rcp-detail. [Último acceso: 12 Septiembre 2015].

[19] Wikipedia, «Información sobre PRIME (Power Line Carrier),» [En línea]. Available: https://en.wikipedia.org/wiki/PRIME_(PLC). [Último acceso: 12 Septiembre 2015].

[20] Cirwatt, «Ficha Técnica del contador Cirwatt B200RCP,» [En línea]. Available: http://circutor.es/docs/FT_Q1_CIRWATT-B200RCP_SP.pdf. [Último acceso: 12 Septiembre 2015].

[21] HP, «Especificaciones del ordenador portátil HP ProBook 4520s,» [En línea]. Available: http://h10010.www1.hp.com/wwpc/pr/es/sm/WF06a/321957-321957-64295-3955552-3955552-4145198.html?dnr=2. [Último acceso: 12 Septiembre 2015].

[22] Wikipedia, «Información sobre OVM,» [En línea]. Available: https://es.wikipedia.org/wiki/Operador_m%C3%B3vil_virtual. [Último acceso: 15 Septiembre 2015].

[23] ITU, «Recomendación de la ITU sobre los comandos AT,» [En línea]. Available: http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-V.250-200307-I!!PDF-E&type=items. [Último acceso: 13 Septiembre 2015].

[24] ETSI, «Estándar para comandos AT referentes a GSM ME,» [En línea]. Available: http://www.etsi.org/deliver/etsi_gts/07/0707/05.00.00_60/gsmts_0707v050000p.pdf. [Último acceso: 13 Septiembre 2015].

[25] ETSI, «High level descriptcion AT Commands - ETSI,» [En línea]. Available: http://www.etsi.org/deliver/etsi_tr/102100_102199/102179/01.01.01_60/tr_102179v010101p.pdf. [Último acceso: 13 Septiembre 2015].

[26] Wikipedia, «Información de los comandos Hayes,» [En línea]. Available: https://es.wikipedia.org/wiki/Conjunto_de_comandos_Hayes. [Último acceso: 13 Septiembre 2015].

[27] Hayes Communications, «Página de Hayes Communications,» [En línea]. Available: http://www.hayescom.com/. [Último acceso: 13 Septriembre 2015].

[28] Wikipedia, «Hayes Command Set,» [En línea]. Available: https://en.wikipedia.org/wiki/Hayes_command_set. [Último acceso: 13 Septiembre 2015].

[29] Oracle, «Especificación de Java - Oracle,» [En línea]. Available: http://docs.oracle.com/javase/specs/. [Último acceso: 13 Septiembre 2015].

[30] Oracle, «Referencia a WORA,» [En línea]. Available: http://www.oracle.com/us/technologies/java/oracle-javase-

embedded-ds-365290.pdf. [Último acceso: 13 Septiembre 2015].

[31] Wikipedia, «Referencia de Java,» [En línea]. Available: https://en.wikipedia.org/wiki/Java_(programming_language). [Último acceso: 13 Septiembre 2015].

[32] Oracle, «Java Compiler - Oracle,» [En línea]. Available: http://www.oracle.com/technetwork/articles/javase/javac-137034.html. [Último acceso: 13 Septiembre 2015].

[33] Oracle, «Javadoc Tool - Oracle,» [En línea]. Available: http://www.oracle.com/technetwork/articles/java/index-jsp-135444.html. [Último acceso: 13 Septiembre 2015].

[34] Oracle, «Página de Java ME,» [En línea]. Available: http://www.oracle.com/technetwork/java/embedded/javame/index.html. [Último acceso: 13 Septiembre 2015].

[35] Oracle, «Información sobre CLDC,» [En línea]. Available: http://www.oracle.com/technetwork/java/cldc-141990.html. [Último acceso: 12 Septiembre 2015].

[36] Oracle, «Información IMP-NG,» [En línea]. Available: https://docs.oracle.com/javame/dev-tools/jme-sdk-3.2/dev-guide/platforms.htm#CHDJFIJI. [Último acceso: 12 Septiembre 2015].

[37] EIA, *Estándar RS232*.

[38] Biblioteca US, «The basics of Recommended Standard 232 (RS-232),» 1997.

[39] E. W. J. P. D. R. Steve Mackay, «3: EIA-232 overview,» de *In Practical Industrial Data Networks 2003:32-52*, Elsevier Ltd, 2004.

[40] «4a: RS232 overview,» de *In Practical Industrial Data Communications 2004:71-81*, Elsevier Ltd, 2004.

[41] IETF, «Transmission Control Protocol - RFC 793,» [En línea]. Available: https://tools.ietf.org/html/rfc793. [Último acceso: 13 Septiembre 2015].

[42] IETF, «HyperText Transfer Protocol - RFC 2616,» [En línea]. Available: https://www.ietf.org/rfc/rfc2616.txt. [Último acceso: 13 Septiembre 2015].

[43] IETF, «Uniform Resource Identifier - RFC 3986,» [En línea]. Available: https://www.ietf.org/rfc/rfc3986.txt. [Último acceso: 13 Septiembre 2015].

[44] IETF, «Simple Mail Transfer Protocol - RFC 821,» [En línea]. Available: https://tools.ietf.org/html/rfc821. [Último acceso: 13 Septiembre 2015].

[45] IETF, «File Transfer Protocol - RFC 959,» [En línea]. Available: https://www.ietf.org/rfc/rfc959.txt. [Último acceso: 13 Septiembre 2015].

[46] Modbus, «Modbus over serial line v1.02 - Documentation,» [En línea]. Available: http://www.modbus.org/docs/Modbus_over_serial_line_V1_02.pdf. [Último acceso: 19 Septiembre 2015].

[47] Eclipse, «Eclipse Helios SR2 versión 3.6.2,» [En línea]. Available: http://www.eclipse.org/downloads/packages/release/Helios/SR2. [Último acceso: 13 Septiembre 2015].

[48] Wikipedia, «Información sobre Eclipse,» [En línea]. Available:

https://es.wikipedia.org/wiki/Eclipse_(software)#Caracter.C3.ADsticas. [Último acceso: 12 Septiembre 2015].

[49] Oracle, «Java SE con JDK versión 6.25,» [En línea]. Available: http://www.oracle.com/technetwork/java/javase/downloads/jre-6u25-download-346243.html. [Último acceso: 13 Septiembre 2015].

[50] Oracle, «Diferencias entre JDK y JRE,» [En línea]. Available: https://www.java.com/es/download/faq/techinfo.xml. [Último acceso: 12 Septiembre 2015].

[51] Putty, «Web de Putty,» [En línea]. Available: http://www.putty.org/. [Último acceso: 12 Septiembre 2015].

[52] Minicom, «Información y ayuda de Minicom,» [En línea]. Available: https://help.ubuntu.com/community/Minicom. [Último acceso: 13 Septiembre 2015].

[53] Apache, «Información sobre Apache HTTP Server Project,» [En línea]. Available: http://httpd.apache.org/ABOUT_APACHE.html. [Último acceso: 13 Septiembre 2015].

[54] PHP, «Página de PHP,» [En línea]. Available: http://php.net/. [Último acceso: 13 Septiembre 2015].

[55] Cinterion, *Java User's Guide (Manual de instalación de CMTK) PDF del CD de instalación,* 2012.

[56] Stack Overflow, «How to set variable JAVA_HOME in Windows 7,» [En línea]. Available: http://stackoverflow.com/questions/2619584/how-to-set-java-home-on-windows-7. [Último acceso: 13 Septiembre 2015].

[57] Oracle, «Mobile Tools for Java,» [En línea]. Available: https://projects.eclipse.org/projects/tools.sequoyah.mtj%20. [Último acceso: 16 Septiembre 2015].

[58] Cinterion, *Java User Guide v19,* 2012.

[59] Gemalto, «Página de Gemalto,» [En línea]. Available: http://www.gemalto.com/. [Último acceso: 13 Septiembre 2015].

[60] JavaCint, «FAQ de JavaCint - Firmware,» [En línea]. Available: http://www.javacint.com/TC65Dev#Firmware. [Último acceso: 13 Septiembre 2015].

[61] Matrix Electrónica, [En línea]. Available: ftp://ftp.matrix.es/mtxm2m/Modems%20M2M/MTX-65i%20Family/MTX-65i%20Family%20User%20Manual.pdf. [Último acceso: 13 Septiembre 2015].

[62] Linux, «Manual de Linux - Comando dmesg,» [En línea]. Available: http://man7.org/linux/man-pages/man1/dmesg.1.html. [Último acceso: 13 Septiembre 2015].

[63] ITU-T, «Recomendación E.212 - Plan de identificación internacional para las redes públicas y los abonos,» [En línea]. Available: https://www.itu.int/rec/T-REC-E.212/es. [Último acceso: 15 Septiembre 2015].

[64] Gemalto, «How to validate a GPRS Connection,» [En línea]. Available: https://developer.gemalto.com/threads/how-validate-if-device-connected-gprs. [Último acceso: 15 Septiembre 2015].

[65] Wavecom, «Comandos AT para GPRS,» [En línea]. Available: http://www.sendsms.cn/download/AT%20Commands%20for%20GPRS_1.3.pdf. [Último acceso: 15 Septiembre 2015].

[66] Honneywell - Scanning and Mobility, «How to set the allowed frequency,» [En línea]. Available:

http://hsm.force.com/publickb/articles/HSM_Article/How-to-set-the-Preferred-and-Allowed-GSM-Bands-in-Windows-Mobile-6. [Último acceso: 15 Septiembre 2015].

[67] Oracle, «MIDlet Class,» [En línea]. Available: http://docs.oracle.com/javame/config/cldc/ref-impl/midp2.0/jsr118/javax/microedition/midlet/MIDlet.html. [Último acceso: 16 Septiembre 2015].

[68] Oracle, «Mobile Information Device Profile (MIDP) - Oracle,» [En línea]. Available: http://www.oracle.com/technetwork/java/index-jsp-138820.html. [Último acceso: 17 Septiembre 2015].

[69] TC65i, *AT Command Set*, 2012.

[70] TC65i, *Java User's Guide*, 2012.

[71] Eclipse, «Software Plugin EclipseME 1.6.8,» [En línea]. Available: http://www.easyeclipse.org/site/plugins/eclipseme.html. [Último acceso: 12 Septiembre 2015].

[72] EclipseME, «Página EclipseME,» [En línea]. Available: http://eclipseme.org/index.html. [Último acceso: 12 Septiembre 2015].

[73] EclipseME, «Información sobre EclipseME,» [En línea]. Available: http://eclipseme.org/docs/faq.html. [Último acceso: 12 Septiembre 2015].

[74] IETF, «Multipurpose Internet Mail Extensions - RFC 2045,» [En línea]. Available: https://tools.ietf.org/html/rfc2045. [Último acceso: 13 Septiembre 2015].

[75] REE, «Protocolo de comunicación entre registradores y concentradores de medidas,» [En línea]. Available: http://www.ree.es/sites/default/files/01_ACTIVIDADES/Documentos/Documentacion-Simel/protoc_RMCM10042002.pdf. [Último acceso: 13 Septiembre 2015].

[76] Oracle, «Wireless Toolkit Download,» [En línea]. Available: http://www.oracle.com/technetwork/java/download-135801.html. [Último acceso: 16 Septiembre 2015].