

R&S® ViCom Interface Description for R&S® TSMx and TSMW Receivers Operating Manual ViCom Version 14.65

1505.1329.42 – 09

The Operating Manual describes the following R&S® ViCom Interface Description

Rohde&Schwarz would like to thank the open source community for their valuable contribution to embedded computing.

© 2012 Rohde & Schwarz GmbH & Co. KG

81671 Munich, Germany

Printed in Germany – Subject to change – Data without tolerance limits is not binding.

R&S® is a registered trademark of Rohde & Schwarz GmbH & Co. KG.

Trade names are trademarks of the owners.

The following abbreviations are used throughout this manual:

R&S® ViCom Interface Description is abbreviated as R&S ViCom Interface Description.

Basic Safety Instructions

Always read through and comply with the following safety instructions!




All plants and locations of the Rohde & Schwarz group of companies make every effort to keep the safety standards of our products up to date and to offer our customers the highest possible degree of safety. Our products and the auxiliary equipment they require are designed, built and tested in accordance with the safety standards that apply in each case. Compliance with these standards is continuously monitored by our quality assurance system. The product described here has been designed, built and tested in accordance with the attached EC Certificate of Conformity and has left the manufacturer's plant in a condition fully complying with safety standards. To maintain this condition and to ensure safe operation, you must observe all instructions and warnings provided in this manual. If you have any questions regarding these safety instructions, the Rohde & Schwarz group of companies will be happy to answer them.

Furthermore, it is your responsibility to use the product in an appropriate manner. This product is designed for use solely in industrial and laboratory environments or, if expressly permitted, also in the field and must not be used in any way that may cause personal injury or property damage. You are responsible if the product is used for any purpose other than its designated purpose or in disregard of the manufacturer's instructions. The manufacturer shall assume no responsibility for such use of the product.








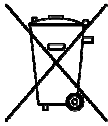

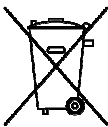

The product is used for its designated purpose if it is used in accordance with its product documentation and within its performance limits (see data sheet, documentation, the following safety instructions). Using the product requires technical skills and, in some cases, a basic knowledge of English. It is therefore essential that only skilled and specialized staff or thoroughly trained personnel with the required skills be allowed to use the product. If personal safety gear is required for using Rohde & Schwarz products, this will be indicated at the appropriate place in the product documentation. Keep the basic safety instructions and the product documentation in a safe place and pass them on to the subsequent users.

Observing the safety instructions will help prevent personal injury or damage of any kind caused by dangerous situations. Therefore, carefully read through and adhere to the following safety instructions before and when using the product. It is also absolutely essential to observe the additional safety instructions on personal safety, for example, that appear in relevant parts of the product documentation. In these safety instructions, the word "product" refers to all merchandise sold and distributed by the Rohde & Schwarz group of companies, including instruments, systems and all accessories. For product-specific information, see the data sheet and the product documentation.

Symbols and safety labels

Symbol	Meaning	Symbol	Meaning
	Notice, general danger location Observe product documentation	○	ON/OFF supply voltage
	Caution when handling heavy equipment	⏻	Standby indication
	Danger of electric shock	— — —	Direct current (DC)

Basic Safety Instructions

Symbol	Meaning	Symbol	Meaning
	Warning! Hot surface		Alternating current (AC)
	Protective conductor terminal		Direct/alternating current (DC/AC)
	Ground		Device fully protected by double (reinforced) insulation
	Ground terminal		EU labeling for batteries and accumulators For additional information, see section "Waste disposal/Environmental protection", item 1.
	Be careful when handling electrostatic sensitive devices		EU labeling for separate collection of electrical and electronic devices For additional information, see section "Waste disposal/Environmental protection", item 2.
	Warning! Laser radiation For additional information, see section "Operation", item 7.		

Signal words and their meaning

The following signal words are used in the product documentation in order to warn the reader about risks and dangers.



Indicates an imminently hazardous situation which, if not avoided, will result in death or serious injury.



Indicates a potentially hazardous situation which, if not avoided, could result in death or serious injury.



Indicates a potentially hazardous situation which, if not avoided, could result in minor or moderate injury.



Indicates the possibility of incorrect operation which can result in damage to the product.

In the product documentation, the word ATTENTION is used synonymously.

These signal words are in accordance with the standard definition for civil applications in the European Economic Area. Definitions that deviate from the standard definition may also exist in other economic areas or military applications. It is therefore essential to make sure that the signal words described here are always used only in connection with the related product documentation and the related product. The use of signal words in connection with unrelated products or documentation can result in misinterpretation and in personal injury or material damage.

Basic Safety Instructions

Operating states and operating positions

The product may be operated only under the operating conditions and in the positions specified by the manufacturer, without the product's ventilation being obstructed. If the manufacturer's specifications are not observed, this can result in electric shock, fire and/or serious personal injury or death. Applicable local or national safety regulations and rules for the prevention of accidents must be observed in all work performed.

1. Unless otherwise specified, the following requirements apply to Rohde & Schwarz products: predefined operating position is always with the housing floor facing down, IP protection 2X, use only indoors, max. operating altitude 2000 m above sea level, max. transport altitude 4500 m above sea level. A tolerance of $\pm 10\%$ shall apply to the nominal voltage and $\pm 5\%$ to the nominal frequency, overvoltage category 2, pollution severity 2.
2. Do not place the product on surfaces, vehicles, cabinets or tables that for reasons of weight or stability are unsuitable for this purpose. Always follow the manufacturer's installation instructions when installing the product and fastening it to objects or structures (e.g. walls and shelves). An installation that is not carried out as described in the product documentation could result in personal injury or even death.
3. Do not place the product on heat-generating devices such as radiators or fan heaters. The ambient temperature must not exceed the maximum temperature specified in the product documentation or in the data sheet. Product overheating can cause electric shock, fire and/or serious personal injury or even death.

Electrical safety

If the information on electrical safety is not observed either at all or to the extent necessary, electric shock, fire and/or serious personal injury or death may occur.

1. Prior to switching on the product, always ensure that the nominal voltage setting on the product matches the nominal voltage of the AC supply network. If a different voltage is to be set, the power fuse of the product may have to be changed accordingly.
2. In the case of products of safety class I with movable power cord and connector, operation is permitted only on sockets with a protective conductor contact and protective conductor.
3. Intentionally breaking the protective conductor either in the feed line or in the product itself is not permitted. Doing so can result in the danger of an electric shock from the product. If extension cords or connector strips are implemented, they must be checked on a regular basis to ensure that they are safe to use.
4. If there is no power switch for disconnecting the product from the AC supply network, or if the power switch is not suitable for this purpose, use the plug of the connecting cable to disconnect the product from the AC supply network. In such cases, always ensure that the power plug is easily reachable and accessible at all times. For example, if the power plug is the disconnecting device, the length of the connecting cable must not exceed 3 m. Functional or electronic switches are not suitable for providing disconnection from the AC supply network. If products without power switches are integrated into racks or systems, the disconnecting device must be provided at the system level.
5. Never use the product if the power cable is damaged. Check the power cables on a regular basis to ensure that they are in proper operating condition. By taking appropriate safety measures and carefully laying the power cable, ensure that the cable cannot be damaged and that no one can be hurt by, for example, tripping over the cable or suffering an electric shock.

Basic Safety Instructions

6. The product may be operated only from TN/TT supply networks fuse-protected with max. 16 A (higher fuse only after consulting with the Rohde & Schwarz group of companies).
7. Do not insert the plug into sockets that are dusty or dirty. Insert the plug firmly and all the way into the socket provided for this purpose. Otherwise, sparks that result in fire and/or injuries may occur.
8. Do not overload any sockets, extension cords or connector strips; doing so can cause fire or electric shocks.
9. For measurements in circuits with voltages $V_{\text{rms}} > 30 \text{ V}$, suitable measures (e.g. appropriate measuring equipment, fuse protection, current limiting, electrical separation, insulation) should be taken to avoid any hazards.
10. Ensure that the connections with information technology equipment, e.g. PCs or other industrial computers, comply with the IEC60950-1/EN60950-1 or IEC61010-1/EN 61010-1 standards that apply in each case.
11. Unless expressly permitted, never remove the cover or any part of the housing while the product is in operation. Doing so will expose circuits and components and can lead to injuries, fire or damage to the product.
12. If a product is to be permanently installed, the connection between the protective conductor terminal on site and the product's protective conductor must be made first before any other connection is made. The product may be installed and connected only by a licensed electrician.
13. For permanently installed equipment without built-in fuses, circuit breakers or similar protective devices, the supply circuit must be fuse-protected in such a way that anyone who has access to the product, as well as the product itself, is adequately protected from injury or damage.
14. Use suitable overvoltage protection to ensure that no overvoltage (such as that caused by a bolt of lightning) can reach the product. Otherwise, the person operating the product will be exposed to the danger of an electric shock.
15. Any object that is not designed to be placed in the openings of the housing must not be used for this purpose. Doing so can cause short circuits inside the product and/or electric shocks, fire or injuries.
16. Unless specified otherwise, products are not liquid-proof (see also section "Operating states and operating positions", item 1). Therefore, the equipment must be protected against penetration by liquids. If the necessary precautions are not taken, the user may suffer electric shock or the product itself may be damaged, which can also lead to personal injury.
17. Never use the product under conditions in which condensation has formed or can form in or on the product, e.g. if the product has been moved from a cold to a warm environment. Penetration by water increases the risk of electric shock.
18. Prior to cleaning the product, disconnect it completely from the power supply (e.g. AC supply network or battery). Use a soft, non-linting cloth to clean the product. Never use chemical cleaning agents such as alcohol, acetone or diluents for cellulose lacquers.

Operation

1. Operating the products requires special training and intense concentration. Make sure that persons who use the products are physically, mentally and emotionally fit enough to do so; otherwise, injuries or material damage may occur. It is the responsibility of the employer/operator to select suitable personnel for operating the products.

Basic Safety Instructions

2. Before you move or transport the product, read and observe the section titled "Transport".
3. As with all industrially manufactured goods, the use of substances that induce an allergic reaction (allergens) such as nickel cannot be generally excluded. If you develop an allergic reaction (such as a skin rash, frequent sneezing, red eyes or respiratory difficulties) when using a Rohde & Schwarz product, consult a physician immediately to determine the cause and to prevent health problems or stress.
4. Before you start processing the product mechanically and/or thermally, or before you take it apart, be sure to read and pay special attention to the section titled "Waste disposal/Environmental protection", item 1.
5. Depending on the function, certain products such as RF radio equipment can produce an elevated level of electromagnetic radiation. Considering that unborn babies require increased protection, pregnant women must be protected by appropriate measures. Persons with pacemakers may also be exposed to risks from electromagnetic radiation. The employer/operator must evaluate workplaces where there is a special risk of exposure to radiation and, if necessary, take measures to avert the potential danger.
6. Should a fire occur, the product may release hazardous substances (gases, fluids, etc.) that can cause health problems. Therefore, suitable measures must be taken, e.g. protective masks and protective clothing must be worn.
7. Laser products are given warning labels that are standardized according to their laser class. Lasers can cause biological harm due to the properties of their radiation and due to their extremely concentrated electromagnetic power. If a laser product (e.g. a CD/DVD drive) is integrated into a Rohde & Schwarz product, absolutely no other settings or functions may be used as described in the product documentation. The objective is to prevent personal injury (e.g. due to laser beams).
8. EMC classes (in line with CISPR 11)
Class A: Equipment suitable for use in all environments except residential environments and environments that are directly connected to a low-voltage supply network that supplies residential buildings.
Class B: Equipment suitable for use in residential environments and environments that are directly connected to a low-voltage supply network that supplies residential buildings.

Repair and service

1. The product may be opened only by authorized, specially trained personnel. Before any work is performed on the product or before the product is opened, it must be disconnected from the AC supply network. Otherwise, personnel will be exposed to the risk of an electric shock.
2. Adjustments, replacement of parts, maintenance and repair may be performed only by electrical experts authorized by Rohde & Schwarz. Only original parts may be used for replacing parts relevant to safety (e.g. power switches, power transformers, fuses). A safety test must always be performed after parts relevant to safety have been replaced (visual inspection, protective conductor test, insulation resistance measurement, leakage current measurement, functional test). This helps ensure the continued safety of the product.

Basic Safety Instructions

Batteries and rechargeable batteries/cells

If the information regarding batteries and rechargeable batteries/cells is not observed either at all or to the extent necessary, product users may be exposed to the risk of explosions, fire and/or serious personal injury, and, in some cases, death. Batteries and rechargeable batteries with alkaline electrolytes (e.g. lithium cells) must be handled in accordance with the EN 62133 standard.

1. Cells must not be taken apart or crushed.
2. Cells or batteries must not be exposed to heat or fire. Storage in direct sunlight must be avoided. Keep cells and batteries clean and dry. Clean soiled connectors using a dry, clean cloth.
3. Cells or batteries must not be short-circuited. Cells or batteries must not be stored in a box or in a drawer where they can short-circuit each other, or where they can be short-circuited by other conductive materials. Cells and batteries must not be removed from their original packaging until they are ready to be used.
4. Cells and batteries must not be exposed to any mechanical shocks that are stronger than permitted.
5. If a cell develops a leak, the fluid must not be allowed to come into contact with the skin or eyes. If contact occurs, wash the affected area with plenty of water and seek medical aid.
6. Improperly replacing or charging cells or batteries that contain alkaline electrolytes (e.g. lithium cells) can cause explosions. Replace cells or batteries only with the matching Rohde & Schwarz type (see parts list) in order to ensure the safety of the product.
7. Cells and batteries must be recycled and kept separate from residual waste. Rechargeable batteries and normal batteries that contain lead, mercury or cadmium are hazardous waste. Observe the national regulations regarding waste disposal and recycling.

Transport

1. The product may be very heavy. Therefore, the product must be handled with care. In some cases, the user may require a suitable means of lifting or moving the product (e.g. with a lift-truck) to avoid back or other physical injuries.
2. Handles on the products are designed exclusively to enable personnel to transport the product. It is therefore not permissible to use handles to fasten the product to or on transport equipment such as cranes, fork lifts, wagons, etc. The user is responsible for securely fastening the products to or on the means of transport or lifting. Observe the safety regulations of the manufacturer of the means of transport or lifting. Noncompliance can result in personal injury or material damage.
3. If you use the product in a vehicle, it is the sole responsibility of the driver to drive the vehicle safely and properly. The manufacturer assumes no responsibility for accidents or collisions. Never use the product in a moving vehicle if doing so could distract the driver of the vehicle. Adequately secure the product in the vehicle to prevent injuries or other damage in the event of an accident.

Waste disposal/Environmental protection

1. Specially marked equipment has a battery or accumulator that must not be disposed of with unsorted municipal waste, but must be collected separately. It may only be disposed of at a suitable collection point or via a Rohde & Schwarz customer service center.

Instrucciones de seguridad elementales

2. Waste electrical and electronic equipment must not be disposed of with unsorted municipal waste, but must be collected separately.
Rohde & Schwarz GmbH & Co. KG has developed a disposal concept and takes full responsibility for take-back obligations and disposal obligations for manufacturers within the EU. Contact your Rohde & Schwarz customer service center for environmentally responsible disposal of the product.
3. If products or their components are mechanically and/or thermally processed in a manner that goes beyond their intended use, hazardous substances (heavy-metal dust such as lead, beryllium, nickel) may be released. For this reason, the product may only be disassembled by specially trained personnel. Improper disassembly may be hazardous to your health. National waste disposal regulations must be observed.
4. If handling the product releases hazardous substances or fuels that must be disposed of in a special way, e.g. coolants or engine oils that must be replenished regularly, the safety instructions of the manufacturer of the hazardous substances or fuels and the applicable regional waste disposal regulations must be observed. Also observe the relevant safety instructions in the product documentation. The improper disposal of hazardous substances or fuels can cause health problems and lead to environmental damage.

For additional information about environmental protection, visit the Rohde & Schwarz website.

Instrucciones de seguridad elementales

¡Es imprescindible leer y cumplir las siguientes instrucciones e informaciones de seguridad!

El principio del grupo de empresas Rohde & Schwarz consiste en tener nuestros productos siempre al día con los estándares de seguridad y de ofrecer a nuestros clientes el máximo grado de seguridad. Nuestros productos y todos los equipos adicionales son siempre fabricados y examinados según las normas de seguridad vigentes. Nuestro sistema de garantía de calidad controla constantemente que sean cumplidas estas normas. El presente producto ha sido fabricado y examinado según el certificado de conformidad adjunto de la UE y ha salido de nuestra planta en estado impecable según los estándares técnicos de seguridad. Para poder preservar este estado y garantizar un funcionamiento libre de peligros, el usuario deberá atenerse a todas las indicaciones, informaciones de seguridad y notas de alerta. El grupo de empresas Rohde & Schwarz está siempre a su disposición en caso de que tengan preguntas referentes a estas informaciones de seguridad.















Además queda en la responsabilidad del usuario utilizar el producto en la forma debida. Este producto está destinado exclusivamente al uso en la industria y el laboratorio o, si ha sido expresamente autorizado, para aplicaciones de campo y de ninguna manera deberá ser utilizado de modo que alguna persona/cosa pueda sufrir daño. El uso del producto fuera de sus fines definidos o sin tener en cuenta las instrucciones del fabricante queda en la responsabilidad del usuario. El fabricante no se hace en ninguna forma responsable de consecuencias a causa del mal uso del producto.

Instrucciones de seguridad elementales


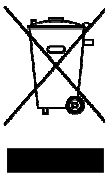

Se parte del uso correcto del producto para los fines definidos si el producto es utilizado conforme a las indicaciones de la correspondiente documentación del producto y dentro del margen de rendimiento definido (ver hoja de datos, documentación, informaciones de seguridad que siguen). El uso del producto hace necesarios conocimientos técnicos y ciertos conocimientos del idioma inglés. Por eso se debe tener en cuenta que el producto solo pueda ser operado por personal especializado o personas instruidas en profundidad con las capacidades correspondientes. Si fuera necesaria indumentaria de seguridad para el uso de productos de Rohde & Schwarz, encontraría la información debida en la documentación del producto en el capítulo correspondiente. Guarde bien las informaciones de seguridad elementales, así como la documentación del producto, y entréguelas a usuarios posteriores.

Tener en cuenta las informaciones de seguridad sirve para evitar en lo posible lesiones o daños por peligros de toda clase. Por eso es imprescindible leer detalladamente y comprender por completo las siguientes informaciones de seguridad antes de usar el producto, y respetarlas durante el uso del producto. Deberán tenerse en cuenta todas las demás informaciones de seguridad, como p. ej. las referentes a la protección de personas, que encontrarán en el capítulo correspondiente de la documentación del producto y que también son de obligado cumplimiento. En las presentes informaciones de seguridad se recogen todos los objetos que distribuye el grupo de empresas Rohde & Schwarz bajo la denominación de "producto", entre ellos también aparatos, instalaciones así como toda clase de accesorios. Los datos específicos del producto figuran en la hoja de datos y en la documentación del producto.

Símbolos y definiciones de seguridad

Símbolo	Significado	Símbolo	Significado
	Aviso: punto de peligro general Observar la documentación del producto		Tensión de alimentación de PUESTA EN MARCHA / PARADA
	Atención en el manejo de dispositivos de peso elevado		Indicación de estado de espera (standby)
	Peligro de choque eléctrico		Corriente continua (DC)
	Advertencia: superficie caliente		Corriente alterna (AC)
	Conexión a conductor de protección		Corriente continua / Corriente alterna (DC/AC)
	Conexión a tierra		El aparato está protegido en su totalidad por un aislamiento doble (reforzado)
	Conexión a masa		Distintivo de la UE para baterías y acumuladores Más información en la sección "Eliminación/protección del medio ambiente", punto 1.

Instrucciones de seguridad elementales

Símbolo	Significado	Símbolo	Significado
	Aviso: Cuidado en el manejo de dispositivos sensibles a la electrostática (ESD)		Distintivo de la UE para la eliminación por separado de dispositivos eléctricos y electrónicos Más información en la sección "Eliminación/protección del medio ambiente", punto 2.
	Advertencia: rayo láser Más información en la sección "Funcionamiento", punto 7.		

Palabras de señal y su significado

En la documentación del producto se utilizan las siguientes palabras de señal con el fin de advertir contra riesgos y peligros.



PELIGRO identifica un peligro inminente con riesgo elevado que provocará muerte o lesiones graves si no se evita.



ADVERTENCIA identifica un posible peligro con riesgo medio de provocar muerte o lesiones (graves) si no se evita.



ATENCIÓN identifica un peligro con riesgo reducido de provocar lesiones leves o moderadas si no se evita.



AVISO indica la posibilidad de utilizar mal el producto y, como consecuencia, dañarlo.

En la documentación del producto se emplea de forma sinónima el término CUIDADO.

Las palabras de señal corresponden a la definición habitual para aplicaciones civiles en el área económica europea. Pueden existir definiciones diferentes a esta definición en otras áreas económicas o en aplicaciones militares. Por eso se deberá tener en cuenta que las palabras de señal aquí descritas sean utilizadas siempre solamente en combinación con la correspondiente documentación del producto y solamente en combinación con el producto correspondiente. La utilización de las palabras de señal en combinación con productos o documentaciones que no les correspondan puede llevar a interpretaciones equivocadas y tener por consecuencia daños en personas u objetos.

Estados operativos y posiciones de funcionamiento

El producto solamente debe ser utilizado según lo indicado por el fabricante respecto a los estados operativos y posiciones de funcionamiento sin que se obstruya la ventilación. Si no se siguen las indicaciones del fabricante, pueden producirse choques eléctricos, incendios y/o lesiones graves con posible consecuencia de muerte. En todos los trabajos deberán ser tenidas en cuenta las normas nacionales y locales de seguridad del trabajo y de prevención de accidentes.

Instrucciones de seguridad elementales

1. Si no se convino de otra manera, es para los productos Rohde & Schwarz válido lo que sigue: como posición de funcionamiento se define por principio la posición con el suelo de la caja para abajo, modo de protección IP 2X, uso solamente en estancias interiores, utilización hasta 2000 m sobre el nivel del mar, transporte hasta 4500 m sobre el nivel del mar. Se aplicará una tolerancia de $\pm 10\%$ sobre el voltaje nominal y de $\pm 5\%$ sobre la frecuencia nominal. Categoría de sobrecarga eléctrica 2, índice de suciedad 2.
2. No sitúe el producto encima de superficies, vehículos, estantes o mesas, que por sus características de peso o de estabilidad no sean aptos para él. Siga siempre las instrucciones de instalación del fabricante cuando instale y asegure el producto en objetos o estructuras (p. ej. paredes y estantes). Si se realiza la instalación de modo distinto al indicado en la documentación del producto, se pueden causar lesiones o, en determinadas circunstancias, incluso la muerte.
3. No ponga el producto sobre aparatos que generen calor (p. ej. radiadores o calefactores). La temperatura ambiente no debe superar la temperatura máxima especificada en la documentación del producto o en la hoja de datos. En caso de sobrecalentamiento del producto, pueden producirse choques eléctricos, incendios y/o lesiones graves con posible consecuencia de muerte.

Seguridad eléctrica

Si no se siguen (o se siguen de modo insuficiente) las indicaciones del fabricante en cuanto a seguridad eléctrica, pueden producirse choques eléctricos, incendios y/o lesiones graves con posible consecuencia de muerte.

1. Antes de la puesta en marcha del producto se deberá comprobar siempre que la tensión preseleccionada en el producto coincida con la de la red de alimentación eléctrica. Si es necesario modificar el ajuste de tensión, también se deberán cambiar en caso dado los fusibles correspondientes del producto.
2. Los productos de la clase de protección I con alimentación móvil y enchufe individual solamente podrán enchufarse a tomas de corriente con contacto de seguridad y con conductor de protección conectado.
3. Queda prohibida la interrupción intencionada del conductor de protección, tanto en la toma de corriente como en el mismo producto. La interrupción puede tener como consecuencia el riesgo de que el producto sea fuente de choques eléctricos. Si se utilizan cables alargadores o regletas de enchufe, deberá garantizarse la realización de un examen regular de los mismos en cuanto a su estado técnico de seguridad.
4. Si el producto no está equipado con un interruptor para desconectarlo de la red, o bien si el interruptor existente no resulta apropiado para la desconexión de la red, el enchufe del cable de conexión se deberá considerar como un dispositivo de desconexión. El dispositivo de desconexión se debe poder alcanzar fácilmente y debe estar siempre bien accesible. Si, p. ej., el enchufe de conexión a la red es el dispositivo de desconexión, la longitud del cable de conexión no debe superar 3 m). Los interruptores selectores o electrónicos no son aptos para el corte de la red eléctrica. Si se integran productos sin interruptor en bastidores o instalaciones, se deberá colocar el interruptor en el nivel de la instalación.
5. No utilice nunca el producto si está dañado el cable de conexión a red. Compruebe regularmente el correcto estado de los cables de conexión a red. Asegúrese, mediante las medidas de protección y de instalación adecuadas, de que el cable de conexión a red no pueda ser dañado o de que nadie pueda ser dañado por él, p. ej. al tropezar o por un choque eléctrico.

Instrucciones de seguridad elementales

6. Solamente está permitido el funcionamiento en redes de alimentación TN/TT aseguradas con fusibles de 16 A como máximo (utilización de fusibles de mayor amperaje solo previa consulta con el grupo de empresas Rohde & Schwarz).
7. Nunca conecte el enchufe en tomas de corriente sucias o llenas de polvo. Introduzca el enchufe por completo y fuertemente en la toma de corriente. La no observación de estas medidas puede provocar chispas, fuego y/o lesiones.
8. No sobrecargue las tomas de corriente, los cables alargadores o las regletas de enchufe ya que esto podría causar fuego o choques eléctricos.
9. En las mediciones en circuitos de corriente con una tensión $U_{\text{eff}} > 30 \text{ V}$ se deberán tomar las medidas apropiadas para impedir cualquier peligro (p. ej. medios de medición adecuados, seguros, limitación de tensión, corte protector, aislamiento etc.).
10. Para la conexión con dispositivos informáticos como un PC o un ordenador industrial, debe comprobarse que éstos cumplan los estándares IEC60950-1/EN60950-1 o IEC61010-1/EN 61010-1 válidos en cada caso.
11. A menos que esté permitido expresamente, no retire nunca la tapa ni componentes de la carcasa mientras el producto esté en servicio. Esto pone a descubierto los cables y componentes eléctricos y puede causar lesiones, fuego o daños en el producto.
12. Si un producto se instala en un lugar fijo, se deberá primero conectar el conductor de protección fijo con el conductor de protección del producto antes de hacer cualquier otra conexión. La instalación y la conexión deberán ser efectuadas por un electricista especializado.
13. En el caso de dispositivos fijos que no estén provistos de fusibles, interruptor automático ni otros mecanismos de seguridad similares, el circuito de alimentación debe estar protegido de modo que todas las personas que puedan acceder al producto, así como el producto mismo, estén a salvo de posibles daños.
14. Todo producto debe estar protegido contra sobretensión (debida p. ej. a una caída del rayo) mediante los correspondientes sistemas de protección. Si no, el personal que lo utilice quedará expuesto al peligro de choque eléctrico.
15. No debe introducirse en los orificios de la caja del aparato ningún objeto que no esté destinado a ello. Esto puede producir cortocircuitos en el producto y/o puede causar choques eléctricos, fuego o lesiones.
16. Salvo indicación contraria, los productos no están impermeabilizados (ver también el capítulo "Estados operativos y posiciones de funcionamiento", punto 1). Por eso es necesario tomar las medidas necesarias para evitar la entrada de líquidos. En caso contrario, existe peligro de choque eléctrico para el usuario o de daños en el producto, que también pueden redundar en peligro para las personas.
17. No utilice el producto en condiciones en las que pueda producirse o ya se hayan producido condensaciones sobre el producto o en el interior de éste, como p. ej. al desplazarlo de un lugar frío a otro caliente. La entrada de agua aumenta el riesgo de choque eléctrico.
18. Antes de la limpieza, desconecte por completo el producto de la alimentación de tensión (p. ej. red de alimentación o batería). Realice la limpieza de los aparatos con un paño suave, que no se deshilache. No utilice bajo ningún concepto productos de limpieza químicos como alcohol, acetona o diluyentes para lacas nitrocelulósicas.

Instrucciones de seguridad elementales

Funcionamiento

1. El uso del producto requiere instrucciones especiales y una alta concentración durante el manejo. Debe asegurarse que las personas que manejen el producto estén a la altura de los requerimientos necesarios en cuanto a aptitudes físicas, psíquicas y emocionales, ya que de otra manera no se pueden excluir lesiones o daños de objetos. El empresario u operador es responsable de seleccionar el personal usuario apto para el manejo del producto.
2. Antes de desplazar o transportar el producto, lea y tenga en cuenta el capítulo "Transporte".
3. Como con todo producto de fabricación industrial no puede quedar excluida en general la posibilidad de que se produzcan alergias provocadas por algunos materiales empleados —los llamados alérgenos (p. ej. el níquel)—. Si durante el manejo de productos Rohde & Schwarz se producen reacciones alérgicas, como p. ej. irritaciones cutáneas, estornudos continuos, enrojecimiento de la conjuntiva o dificultades respiratorias, debe avisarse inmediatamente a un médico para investigar las causas y evitar cualquier molestia o daño a la salud.
4. Antes de la manipulación mecánica y/o térmica o el desmontaje del producto, debe tenerse en cuenta imprescindiblemente el capítulo "Eliminación/protección del medio ambiente", punto 1.
5. Ciertos productos, como p. ej. las instalaciones de radiocomunicación RF, pueden a causa de su función natural, emitir una radiación electromagnética aumentada. Deben tomarse todas las medidas necesarias para la protección de las mujeres embarazadas. También las personas con marcapasos pueden correr peligro a causa de la radiación electromagnética. El empresario/operador tiene la obligación de evaluar y señalar las áreas de trabajo en las que exista un riesgo elevado de exposición a radiaciones.
6. Tenga en cuenta que en caso de incendio pueden desprenderse del producto sustancias tóxicas (gases, líquidos etc.) que pueden generar daños a la salud. Por eso, en caso de incendio deben usarse medidas adecuadas, como p. ej. máscaras antigás e indumentaria de protección.
7. Los productos con láser están provistos de indicaciones de advertencia normalizadas en función de la clase de láser del que se trate. Los rayos láser pueden provocar daños de tipo biológico a causa de las propiedades de su radiación y debido a su concentración extrema de potencia electromagnética. En caso de que un producto Rohde & Schwarz contenga un producto láser (p. ej. un lector de CD/DVD), no debe usarse ninguna otra configuración o función aparte de las descritas en la documentación del producto, a fin de evitar lesiones (p. ej. debidas a irradiación láser).
8. Clases CEM (según CISPR 11)
Clase A: dispositivo apropiado para el uso en cualquier zona excepto en áreas residenciales y en aquellas zonas que se encuentran conectadas a una red de suministro de baja tensión que alimenta un edificio de viviendas.
Clase B: dispositivo apropiado para el uso en áreas residenciales y en aquellas zonas que se encuentran conectadas a una red de suministro de baja tensión que alimenta un edificio de viviendas.

Reparación y mantenimiento

1. El producto solamente debe ser abierto por personal especializado con autorización para ello. Antes de manipular el producto o abrirlo, es obligatorio desconectarlo de la tensión de alimentación, para evitar toda posibilidad de choque eléctrico.

Instrucciones de seguridad elementales

2. El ajuste, el cambio de partes, el mantenimiento y la reparación deberán ser efectuadas solamente por electricistas autorizados por Rohde & Schwarz. Si se reponen partes con importancia para los aspectos de seguridad (p. ej. el enchufe, los transformadores o los fusibles), solamente podrán ser sustituidos por partes originales. Después de cada cambio de partes relevantes para la seguridad deberá realizarse un control de seguridad (control a primera vista, control del conductor de protección, medición de resistencia de aislamiento, medición de la corriente de fuga, control de funcionamiento). Con esto queda garantizada la seguridad del producto.

Baterías y acumuladores o celdas

Si no se siguen (o se siguen de modo insuficiente) las indicaciones en cuanto a las baterías y acumuladores o celdas, pueden producirse explosiones, incendios y/o lesiones graves con posible consecuencia de muerte. El manejo de baterías y acumuladores con electrolitos alcalinos (p. ej. celdas de litio) debe seguir el estándar EN 62133.

1. No deben desmontarse, abrirse ni triturarse las celdas.
2. Las celdas o baterías no deben someterse a calor ni fuego. Debe evitarse el almacenamiento a la luz directa del sol. Las celdas y baterías deben mantenerse limpias y secas. Limpiar las conexiones sucias con un paño seco y limpio.
3. Las celdas o baterías no deben cortocircuitarse. Es peligroso almacenar las celdas o baterías en estuches o cajones en cuyo interior puedan cortocircuitarse por contacto recíproco o por contacto con otros materiales conductores. No deben extraerse las celdas o baterías de sus embalajes originales hasta el momento en que vayan a utilizarse.
4. Las celdas o baterías no deben someterse a impactos mecánicos fuertes indebidos.
5. En caso de falta de estanqueidad de una celda, el líquido vertido no debe entrar en contacto con la piel ni los ojos. Si se produce contacto, lavar con agua abundante la zona afectada y avisar a un médico.
6. En caso de cambio o recarga inadecuados, las celdas o baterías que contienen electrolitos alcalinos (p. ej. las celdas de litio) pueden explotar. Para garantizar la seguridad del producto, las celdas o baterías solo deben ser sustituidas por el tipo Rohde & Schwarz correspondiente (ver lista de recambios).
7. Las baterías y celdas deben reciclarse y no deben tirarse a la basura doméstica. Las baterías o acumuladores que contienen plomo, mercurio o cadmio deben tratarse como residuos especiales. Respete en esta relación las normas nacionales de eliminación y reciclaje.

Transporte

1. El producto puede tener un peso elevado. Por eso es necesario desplazarlo o transportarlo con precaución y, si es necesario, usando un sistema de elevación adecuado (p. ej. una carretilla elevadora), a fin de evitar lesiones en la espalda u otros daños personales.
2. Las asas instaladas en los productos sirven solamente de ayuda para el transporte del producto por personas. Por eso no está permitido utilizar las asas para la sujeción en o sobre medios de transporte como p. ej. grúas, carretillas elevadoras de horquilla, carros etc. Es responsabilidad suya fijar los productos de manera segura a los medios de transporte o elevación. Para evitar daños personales o daños en el producto, siga las instrucciones de seguridad del fabricante del medio de transporte o elevación utilizado.

Instrucciones de seguridad elementales

3. Si se utiliza el producto dentro de un vehículo, recae de manera exclusiva en el conductor la responsabilidad de conducir el vehículo de manera segura y adecuada. El fabricante no asumirá ninguna responsabilidad por accidentes o colisiones. No utilice nunca el producto dentro de un vehículo en movimiento si esto pudiera distraer al conductor. Asegure el producto dentro del vehículo debidamente para evitar, en caso de un accidente, lesiones u otra clase de daños.

Eliminación/protección del medio ambiente

1. Los dispositivos marcados contienen una batería o un acumulador que no se debe desechar con los residuos domésticos sin clasificar, sino que debe ser recogido por separado. La eliminación se debe efectuar exclusivamente a través de un punto de recogida apropiado o del servicio de atención al cliente de Rohde & Schwarz.
2. Los dispositivos eléctricos usados no se deben desechar con los residuos domésticos sin clasificar, sino que deben ser recogidos por separado. Rohde & Schwarz GmbH & Co.KG ha elaborado un concepto de eliminación de residuos y asume plenamente los deberes de recogida y eliminación para los fabricantes dentro de la UE. Para desechar el producto de manera respetuosa con el medio ambiente, diríjase a su servicio de atención al cliente de Rohde & Schwarz.
3. Si se trabaja de manera mecánica y/o térmica cualquier producto o componente más allá del funcionamiento previsto, pueden liberarse sustancias peligrosas (povos con contenido de metales pesados como p. ej. plomo, berilio o níquel). Por eso el producto solo debe ser desmontado por personal especializado con formación adecuada. Un desmontaje inadecuado puede ocasionar daños para la salud. Se deben tener en cuenta las directivas nacionales referentes a la eliminación de residuos.
4. En caso de que durante el trato del producto se formen sustancias peligrosas o combustibles que deban tratarse como residuos especiales (p. ej. refrigerantes o aceites de motor con intervalos de cambio definidos), deben tenerse en cuenta las indicaciones de seguridad del fabricante de dichas sustancias y las normas regionales de eliminación de residuos. Tenga en cuenta también en caso necesario las indicaciones de seguridad especiales contenidas en la documentación del producto. La eliminación incorrecta de sustancias peligrosas o combustibles puede causar daños a la salud o daños al medio ambiente.

Se puede encontrar más información sobre la protección del medio ambiente en la página web de Rohde & Schwarz.

Qualitätszertifikat

Certificate of quality

Certificat de qualité

Certified Quality System
ISO 9001

Certified Environmental System
ISO 14001

Sehr geehrter Kunde,

Sie haben sich für den Kauf eines Rohde&Schwarz-Produktes entschieden. Sie erhalten damit ein nach modernsten Fertigungsverfahren hergestelltes Produkt. Es wurde nach den Regeln unseres Qualitätsmanagementsystems entwickelt, gefertigt und geprüft. Das Rohde&Schwarz-Qualitätsmanagementsystem ist unter anderem nach ISO9001 und ISO14001 zertifiziert.

Der Umwelt verpflichtet

- Energie-effiziente, RoHS-konforme Produkte
- Kontinuierliche Weiterentwicklung nachhaltiger Umweltkonzepte
- ISO 14001-zertifiziertes Umweltmanagementsystem

Dear customer,

You have decided to buy a Rohde&Schwarz product. You are thus assured of receiving a product that is manufactured using the most modern methods available. This product was developed, manufactured and tested in compliance with our quality management system standards. The Rohde&Schwarz quality management system is certified according to standards such as ISO9001 and ISO14001.

Environmental commitment

- Energy-efficient products
- Continuous improvement in environmental sustainability
- ISO 14001-certified environmental management system

Cher client,

Vous avez choisi d'acheter un produit Rohde&Schwarz. Vous disposez donc d'un produit fabriqué d'après les méthodes les plus avancées. Le développement, la fabrication et les tests respectent nos normes de gestion qualité. Le système de gestion qualité de Rohde&Schwarz a été homologué, entre autres, conformément aux normes ISO9001 et ISO14001.

Engagement écologique

- Produits à efficience énergétique
- Amélioration continue de la durabilité environnementale
- Système de gestion de l'environnement certifié selon ISO 14001



Customer Support

Technical support – where and when you need it

For quick, expert help with any Rohde & Schwarz equipment, contact one of our Customer Support Centers. A team of highly qualified engineers provides telephone support and will work with you to find a solution to your query on any aspect of the operation, programming or applications of Rohde & Schwarz equipment.

Up-to-date information and upgrades

To keep your instrument up-to-date and to be informed about new application notes related to your instrument, please send an e-mail to the Customer Support Center stating your instrument and your wish. We will take care that you will get the right information.

Europe, Africa, Middle East

Phone +49 89 4129 12345
customersupport@rohde-schwarz.com

North America

Phone 1-888-TEST-RSA (1-888-837-8772)
customer.support@rsa.rohde-schwarz.com

Latin America

Phone +1-410-910-7988
customersupport.la@rohde-schwarz.com

Asia/Pacific

Phone +65 65 13 04 88
customersupport.asia@rohde-schwarz.com

China

Phone +86-800-810-8228 /
+86-400-650-5896
customersupport.china@rohde-schwarz.com



Table of Contents

1	Important Notes	8
1.1	References.....	8
1.2	Key to Technical terms	8
2	General Description.....	10
2.1	TSMx Product Family	11
2.1.1	Technology specific features	12
3	Getting Started.....	14
3.1	What you need	14
3.1.1	Hardware Requirements.....	14
3.1.2	Software Requirements	15
3.1.3	Installing the ViCom Interface Dataset and Demo Applications	16
3.2	Installing the ROMES Demo Application.....	20
3.3	Starting the ROMES Demo Application	20
3.4	Prerequisites if ROMES Demo Application has not been Installed.....	21
3.5	Diagnostics Information.....	22
3.5.1	RS232 output with the TSMx.....	22
3.5.2	Managing more than one R&S TSMx scanner	23
4	Programming with the ViCom Interface.....	24
4.1	Device Details.....	24
4.1.1	Measurement Scheduling on the TSMx	24
4.1.2	Resource Allocation on the TSMW.....	25
4.2	Start Programming	27
4.2.1	Prerequisites	27
4.2.2	On Startup.....	28
4.2.3	R&S TSMx Management Functions	29
4.2.4	Using the ViCom data structures	30
4.2.5	Reading and Changing Settings	30
4.2.6	Getting Measurements	30
4.2.7	Releasing the Scanner	31

4.3	Differences between TSMx and TSMW.....	31
4.4	Using the Demodulators	33
4.5	Setting up a Custom Project.....	36
4.5.1	Project Structure	36
4.5.2	Create the Project Environment.....	37
4.5.3	Project Settings.....	37
4.5.4	Working with the Code.....	38
4.5.5	Installation Issues	39
4.5.6	Updating the projects.....	40
4.6	Debugging and Error Handling	40
4.6.1	Debugging Techniques	40
4.6.2	Message Handler.....	41
5	ViCom WCDMA PN Scans.....	43
5.1	Measuring with WCDMA PN Scanner	43
5.1.1	CPICH Channel Impulse Response (CIR) Measurements.....	44
5.1.2	Peak Information.....	45
5.2	Sample Application.....	45
5.2.1	Setup connection	47
5.2.2	Setup a measurement command.....	47
5.2.3	Save and Load.....	49
5.2.4	Update GUI with current scanner settings.....	50
5.2.5	Miscellaneous	50
5.3	BCH Demodulation	51
5.3.1	Measurement Details.....	51
5.3.2	Sample Application	55
5.3.3	Programming Sample	59
5.3.4	Frequently Asked Question	65
6	ViCom GSM Network Scanner	67
6.1	Measuring GSM Signals.....	67
6.1.1	Network Scanner Specials.....	67
6.1.2	Measurement Tasks	69
6.2	The GSM Network Scanner Demo Application	80
6.2.1	Basic Channel Configuration	81

6.2.2	Specifying Measurement Details	82
6.2.3	Requesting System Type Information.....	84
6.3	The Sample.....	85
6.4	GSM BCH Demodulation.....	91
6.4.1	Sample Application	91
6.4.2	Code Listing.....	93
7	ViCom CDMA 2000 Scanner.....	98
7.1	Doing Measurements	98
7.1.1	Measurements	98
7.1.2	Configuration.....	102
7.2	Sample Application.....	104
7.3	Sample Code	106
7.4	CDMA 2000 BCH Demodulation	112
7.4.1	Measurement Algorithm.....	113
7.4.2	Sample Application	113
7.4.3	Code Example	114
7.5	EVDO Measurements	118
7.5.1	The SEvdoControlSettings structure	119
7.6	EVDO BCH Demodulation.....	120
8	LTE Measurements.....	121
8.1	Hardware.....	121
8.2	Measuring LTE	122
8.2.1	Configuration.....	122
8.2.2	Measurement Result.....	126
8.2.3	Error Handling.....	127
8.3	BCH Demodulation	127
8.4	GUI Sample Application	128
8.4.1	Using the Sample Application.....	128
8.4.2	Code analysis	135
8.5	Command Line Sample Application	136
9	WiMAX Measurements	147
9.1	Measuring WiMAX.....	147

9.1.1	Configuration.....	148
9.1.2	Measurement Result.....	149
9.1.3	Error Handling.....	152
9.2	GUI Sample Application	152
9.2.1	Setup connection	153
9.2.2	Measurement controls	154
9.2.3	View Results	156
9.2.4	Update GUI with current scanner settings.....	157
9.2.5	Miscellaneous	157
9.2.6	Demodulation.....	158
9.3	Command Line Sample Application	159
9.3.1	Initialization Sequence	173
9.3.2	Measurement Configuration	174
9.3.3	Results Output	174
9.3.4	Command Line Arguments	174
10	ViCom RF Power Scans	175
10.1	Measuring with the RF Power Scanner	175
10.2	Architecture and Functionality of the RF Power Scanner	177
10.2.1	System Layers	177
10.2.2	The Post Processor Chain.....	178
10.3	Sample Application.....	187
10.3.1	A Walk-Through Example.....	188
10.3.2	Sample Project Organization	192
10.4	Sample	193
10.5	RF Power Scan with TSMW	197
10.5.1	Measurement and Post-Processing Concept	197
10.5.2	Interface Concept.....	197
10.5.3	Sample application.....	198
10.6	RF Power Scanner Specific Trouble Shooting	200
11	CW Measurements.....	201
11.1	Channel Power Measurements.....	201
11.1.1	Measurement Modes	202
11.1.2	Measurement Scheduling.....	204

11.1.3	Channel Power Aggregation.....	208
11.1.4	Sample Application.....	209
11.2	Code Example.....	211
11.3	Error Codes.....	218
12	ViCom GPS.....	219
12.1	Sample Application.....	219
12.2	Measurements.....	220
12.2.1	TSMW Address.....	220
12.2.2	Message format.....	220
12.2.3	Measurement.....	221
12.2.4	Commands.....	221
12.2.5	Ending the session.....	222
12.2.6	Viewing the results.....	222
13	RS232 Tunneling.....	223
13.1	Sample Application.....	223
13.1.1	Walk-Through Example.....	225
13.1.2	Result Loopback to RS232.....	226
13.1.3	Silent Mode.....	226
13.2	Sample Code.....	226
A	TSMW Configuration.....	230
A.1	System Information.....	230
A.2	Option Handling.....	231
A.3	Firmware Update.....	232
B	TSMx Option Handling.....	232
B.1	Prerequisites.....	232
B.2	Program Start.....	233
B.3	Display Contents of the R&S TSMx Info Tab.....	233
B.4	Display Contents of the R&S TSMx Options Tab.....	234
C	Installing the TSMx Windows device driver manually.....	236
D	TSMx Firmware Upgrade.....	238
E	Frequently Asked Questions.....	240
F	UMTS Technical Notes.....	241
F.1	UMTS Channels.....	241

F.2 UTRA operating bands and channel numbers: Downlink.....242

F.3 UTRA operating bands and channel numbers: Uplink243

F.4 System Information Blocks243

Conventions Used in the Documentation

The following conventions are used throughout the R&S ViCom Interface Description Operating Manual:

Typographical conventions

Convention	Description
"Graphical user interface elements"	All names of graphical user interface elements both on the screen and on the front and rear panels, such as dialog boxes, soft keys, menus, options, buttons etc., are enclosed by quotation marks.
"KEYS"	Key names are written in capital letters and enclosed by quotation marks.
<i>Input</i>	Input to be entered by the user is displayed in italics.
File names, commands, program code	File names, commands, coding samples and screen output are distinguished by their font.
"Links"	Links that you can click are displayed in blue font.
"References"	References to other parts of the documentation are enclosed by quotation marks.

Other conventions

- **Remote commands:** Remote commands may include abbreviations to simplify input. In the description of such commands, all parts that have to be entered are written in capital letters. Additional text in lower-case characters is for information only.
- **Procedure descriptions:** When describing how to operate the device, several alternative methods may be available to perform the same task. In this case, the procedure using the touch screen is described, where available. Any elements that can be activated by touching can also be clicked using an additionally connected mouse. The alternative procedure using the keys on the device or the on-screen keyboard is only described if it deviates from the standard operating procedures as described in the Quick Start Guide under "Basic Operations".

The terms "**select**" and "**press**" may refer to any of the described methods, i.e. using a finger on the touch screen, a mouse pointer in the display, or a key on the device or on a keyboard.

1 Important Notes

In this document, the R&S® TSML-x (where x stands for the R&S TSML model) and R&S TSMU/Q/M product series is generally abbreviated as R&S TSMx. Unless explicitly noted, the abbreviation R&S TSMx is valid for all models in the series.

The R&S® TSMW device is always referred to as TSMW. Differences in handling between TSMW and TSMx devices are mentioned in the chapters directly.

1.1 References

The R&S TSMx manuals may be found on the CD-ROM accompanying the instrument.

1. R&S TSML manual
2. R&S TSMU/Q manual
3. R&S TSMW manual
4. R&S ROMES Coverage Measurement Software (Online Help)
5. 3GPP TS 21.905 v5.x.x Technical Terms and Abbreviations
6. 3GPP TS 25.331 v5.x.x Radio Resource Control Protocol Specification
7. 3GPP TS 25.101 v5.x.x User Equipment Radio Transmission and Reception (FDD)
8. 3GPP TS 25.215 v5.x.x Physical Layer Measurements (FDD)

1.2 Key to Technical terms

Abbreviation	Meaning
A-GPS	Assisted GPS
BCD	Binary Coded Decimal
BCH	Broadcast Channel. See Reference 4.
CIR	Channel Impulse Response
C-PICH	See Reference 4.
DLL	Dynamic Linked Library
Firewire	See IEEE1394.
FPGA	Field Programmable Gate Array
GPS	Global Positioning System
IEEE1394	Used interchangeably with Firewire. Fast connection specified by IEEE 1394.
OEM	Original Equipment Manufacturer
OHCI	Open Host Controller Interface

Abbreviation	Meaning
OTDOA	Observed Time Difference Of Arrival
PPS	Pulse Per Second
P-SYNC	Primary Synchronisation code transmitted on the UMTS Primary Synchronization Channel (PSCH)
PSCH	Primary Synchronisation Channel
R&S	Rohde & Schwarz GmbH & Co. KG
R&S TSML-CW	Radio Network Analyzer TSML-CW (RF Continuous Wave measurements (CW))
R&S TSML-G	Radio Network Analyzer TSML-G (GSM)
R&S TSML-C	Radio Network Analyzer TSML-C (IS95/CDMA 2000)
R&S TSML-E	Radio Network Analyzer TSML-E (CDMA2000 EV-DO)
R&S TSML-W	Radio Network Analyzer TSML-W (WCDMA)
R&S TSMQ	Radio Network Analyzer TSMQ (Multi Tech)
R&S TSMU	Radio Network Analyzer TSMU (Single Tech)
R&S TSMW	High-End Radio Network Analyzer TSMW (Multi Tech)
RMS	Root Mean Square
SC	Scrambling Code
SIB	System Information Block
SSCH	Secondary Synchronisation Channel
TDD	Time Division Duplex
TSMx	Signifies any of the TSML-W, TSML-C, TSML-G, TSML-CW, TSML-E, TSML-GW, TSMU or TSMQ scanners.

2 General Description

This document describes the ViCom software interface for Rohde & Schwarz's TSMx scanners. Using the ViCom interface, developers can easily integrate the R&S TSMx receiver types into their own Coverage Measurement Software applications as a kind of OEM receiver. R&S TSMx receiver types offer solutions for the various kinds of mobile radio network standards, e.g. WCDMA, GSM, CDMA2000 or CDMA2000 EV-DO and CW (RF Power Measurement).

For more information about R&S TSMx/TSMW scanners, please see Reference [1]-[3].

The ViCom interface allows you to

- set up a R&S TSMx/TSMW instrument connection
- set up measurement commands
- start and stop measurements from a R&S TSMx/TSMW
- retrieve measurement results
- get error information
- Generate log files.

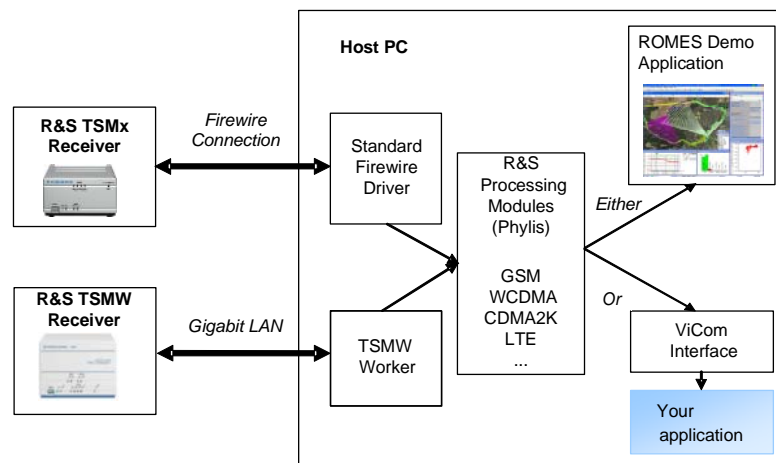


Figure 1 System overview of the software modules on the controlling computer

The control computer is connected to the R&S TSMx receiver by an IEEE1394 (Firewire) connection. A Windows device driver available from Rohde & Schwarz handles communication with the R&S TSMx.

In case of the TSMW, the device is connected using a standard LAN cable. This must support Gigabit LAN and should therefore be at least a Cat5 cable.

The R&S ROMES demo software application is a demonstration version of Rohde & Schwarz's standard Coverage and Measurement Software to interface with the R&S TSMx/TSMW receivers. A ROMES demo application is shipped with every R&S TSMx/TSMW device. It is documented in Reference 4, which can be found on the accompanying CD-ROM.

The ViCom interface includes C++ functions and data structures to control and manage the R&S TSMx scanner. The exact content of the ViCom interface depends upon which type of R&S TSMx scanner is to be used by the application.





A group of basic ViCom functions, that are common to all R&S TSMx receivers, is present in every ViCom interface. Other ViCom functions are R&S TSMx device dependent, and may not be present in every ViCom interface.

R&S TSMx control software includes libraries that are required by the ViCom interface.

2.1 TSMx Product Family

The TSMx product family contains several different devices, each having some significant features making them suitable for different measurement situations. All of these devices which are supported by the ViCom API and the differences are listed in the table shown below.

Table 1 TSMx capabilities matrix

	TSMU 	TSMQ 	TSML-G TSML-C TSML-W	TSML-CW 	TSMW 
Connection	Firewire	Firewire	Firewire	Firewire	LAN
Min. Frequency	80 MHz	80 MHz	80 MHz	80 MHz	30 MHz
Max. Frequency	3 GHz	3 GHz	3 GHz	6 GHz	6 GHz
Resolution Bandwidth	10 kHz	10 kHz	10 kHz	10 kHz	
RAKE Receivers	2500	2500	512 ¹⁾²⁾	-	Limited by SW only
High Speed Mode	-	✓	-	-	-
Parallel Measurements	-	✓	-	-	✓
Max. Measurement Speed (Single Mode)		333 Hz (WCDMA)	40 Hz ¹⁾ 10 Hz ²⁾	625 Hz	
Supported technologies:					
GSM	✓	✓	(✓) ¹⁾	-	✓
WCDMA	✓	✓	(✓) ²⁾	-	✓
CDMA 2000/ Ev-DO	✓	✓	(✓) ³⁾	-	✓
LTE	-	-	-	-	✓
CW	✓	✓	-	✓	(-) ⁴⁾
RF Power Scan	✓	✓	-	-	✓
RS 232	✓	✓	✓	-	-
WIMAX	-	-	-	-	✓

¹⁾ only TSML-G

²⁾ only TSML-W and TSML-C

⁴⁾ supported in upcoming versions

The type of TSMx device can be requested during the usage of the ViCom API and once the initial setup has been performed. The `SConnectedReceiver` structure contains the type descriptor of the connected receiver. It can be requested using the `GetConnectedReceivers()` method of the basic ViCom interface, after the `ViComLoader` has been used to load an interface successfully.

2.1.1 Technology specific features

Some of the functionality that is specified to a certain technology API differs in the same way as described above for different devices.

2.1.1.1 GSM

The GSM Network Scanner has two different settings that depend on the type of the device. These are the measurement rate and the maximum percentage to which the CPU is utilized when the BCH demodulation is performed. In the case of the TSML series the figures are reduced in all cases.

	TSMU	TSMQ	TSML-G	TSMW
Max. Measurement Rate	80 Hz	100 Hz	40 Hz	500 Hz
Max. Load for BCH decoding	50 %	50 %	10 %	50 %

2.1.1.2 WCDMA

The WCDMA changes a whole bunch of constants and thresholds depending on the currently connected device:

- The maximum number of channels that can be measured in an active measurement sequentially
- The measurement rates, where different limits are set for high speed and high dynamic mode. The latter one is only available on TSMQ devices.
- For the demodulation process, a certain minimum E_c/I_0 must be measured to start the algorithms. This limit varies with the device used, as does the maximum amount of CPU time that is allocated for that task in the worst case (see GSM).

	TSMU	TSMQ	TSML-W	TSMW
Max. Number of Channels	32	32	6	32
Max. Measurement Rate in High Dynamic Mode	12 Hz	12 Hz	12 Hz	200 Hz
Max. Measurement Rate in High Speed Mode	20 Hz	50 Hz	10 Hz	200 Hz
Min. E_c/I_0 threshold for demodulation	-20 dB	-25 dB	-10 dB	-25 dB
Max. Load for BCH decoding	50%	50%	10%	50%

2.1.1.3 CDMA 2000

The part of the settings in the CDMA 2000 scanner API that depend on the device type are basically the same as the ones defined for UMTS. The only difference is that there is no High Dynamic and High Speed Mode available, so only one maximum measurement rate is contained in the table below.

	TSMU	TSMQ	TSML-C	TSMW
Max. Number of Channels	32	32	6	32
Max. Measurement Rate	12 Hz	12 Hz	12 Hz	12 Hz
Min. Ec/I0 threshold for demodulation	-20 dB	-25 dB	-10 dB	-25 dB
Max. Load for demodulation	50%	50%	10%	50 %

2.1.1.4 LTE

The LTE scanner does not offer support for TSMx devices, so the device specific parameters table is limited to the TSMW device.

	TSMW
Max. Number of Channels	32
Max. Measurement Rate (blocks per sec)	10 Hz

One block (100 ms) contains up to 20 sync signals => 200 sync signals per sec = 200 Hz

2.1.1.5 WIMAX

The WiMAX scanner does not offer support for TSMx devices, so the device specific parameters table is limited to the TSMW device.

	TSMW
Max. Number of Channels	32
Max. Measurement Rate	14,5 Hz

3 Getting Started

The following applications are provided on the CD-ROM, along with the ViCom interfaces and R&S TSMx control software

- **ROMES Demo Application**, which shows the measurement performance of the R&S TSMx and
- **Programming Demo Application(s)**, which demonstrates the basic functionality of the ViCom interface(s).

The R&S ROMES demo application is a demonstration version of the R&S ROMES radio network analysis software. This software can be used to show the performance of the R&S TSMx scanner, before starting to program your own application. The programming demo applications provide a simple and direct interface to the ViCom functions, without doing any further processing. It gives an example how to program with the ViCom interface.

It is strongly recommended to run both the ROMES demo application and the test application provided, to get familiar with using both the R&S TSMx receiver, and the ViCom interface.

NOTICE

Please note that only one application can be used at once, to control any configuration of R&S TSMx receivers from one computer. For example, it is not possible to run your application, and the test application at the same time, although they may both be installed on the computer.

It is recommended to run the ROMES demo application before running the test application, because the installation process for ROMES demo installs the Windows device driver for the R&S TSMx automatically. If this is not done, a standalone installation of the R&S TSMx device driver is necessary.

3.1 What you need

3.1.1 Hardware Requirements

- Host PC or laptop as follows:
 - running Windows XP, SP2 or newer
 - with at least 2 GHz CPU clock speed, 512 kB cache
 - with 512 MB of RAM
 - with Firewire connection or Ethernet connection, depending on the used scanner model

- R&S Scanner
 - R&S TSMx receiver with OEM option(s) (TSMU-K3x), with
 - power cable for the R&S TSMx (included with the R&S TSMx shipment)
 - a 12 V/1 A DC power supply (not included, for example R&S TSML-Z1, R&S TSMU-Z1)
 - Firewire cable (6-6 pin connector and 6-4 pin connectors are included)
 - Optional: for diagnostics (optional) an RS232 cable, with adapter if necessary, to connect to the control computer (null modem cable, not included)
 - Optional: Synchronizing the R&S TSMx's internal clock from an external source via the 1PPS input connector, an appropriate connection cable/divider is needed that will output 1PPS (not included, e.g. R&S accessory TS-PNSYNC Ident. Nr. 1114.4817.02) (see **Synchronizing the R&S TSMx internal clock from an external source**)
 - R&S TSMW receiver with OEM option(s), with
 - A Gigabit Ethernet (min. Cat5) cable to connect PC and TSMW together
 - Power supply
 - Optionally a GPS antenna (not required for ViCom)
- antenna or standard RF connection to a base station signal generator (partly part of the package, depending of the R&S TSMx/TSMW model)

3.1.1.1 Synchronizing the R&S TSMx internal clock from an external source

The R&S TSMx has an internal clock; however it will normally try to synchronize to an external source, to improve the accuracy of measurements.

If a GSM/UMTS signal is available, the R&S TSMx/TSMW will normally synchronies to this via its antenna. However, if the scanner is being used with a signal generator, or if a high-precision GPS PPS reference is available, the R&S TSMx may be synchronized via the IN PULSE connector on the back panel.

The R&S TSMx accepts an incoming frequency reference of 1 pulse per second (PPS) via the IN PULSE connector. The reference frequency of the signal generator is typically 10 MHz. In such cases a divider is necessary to be connected between the reference output of the signal generator and the PULS IN input of the R&S TSMx.

A divider is available from Rohde & Schwarz, TS-PNSYNC Synchronization Unit for PN Scanner, part number 1114.4817.02.

3.1.2 Software Requirements

The R&S TSMx CD-ROM contains the complete R&S ViCom interface dataset and a demo version of the coverage and network optimization software R&S ROMES.

The R&S ROMES demo software is located in the R&S TSMx CD-ROM subdirectory **..\ROMES_Demo**.

The R&S ViCom interface package is located in the **..\ViCom subfolder**.

The ViCom subfolder contains a setup executable (setup.exe) which installs the ViCom interface development kit on a PC and a copy of this manual.

After running the setup.exe the following components have been installed on the PC ViCom interface files (separated in different technologies)

ViCom sample application(s)

- R&S TSMx IEEE1394 (Firewire) device driver install utility
- R&S TSMx OptionKeyInstaller utility

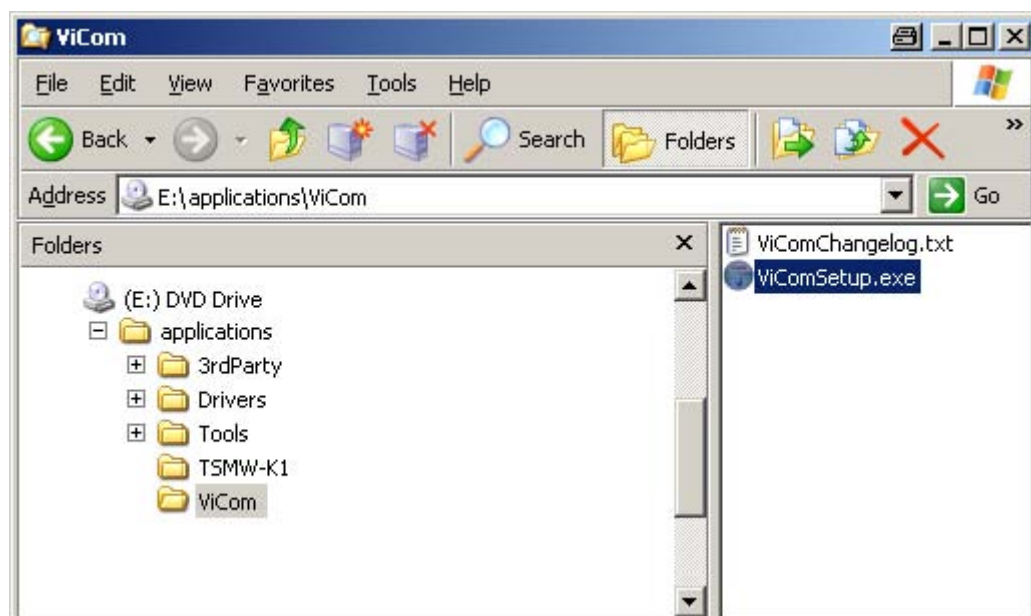
The R&S TSMx ViCom manual and the R&S TSMx utility tools can be accessed via a shortcut in the Windows start menu “**Start->Programs->Rohde & Schwarz ->ViCom**”.

With an internet browser, the internal configuration web page of the TSMW can be viewed using the IP address (default is 192.168.0.2). See Configuration of the TSMW for more details.

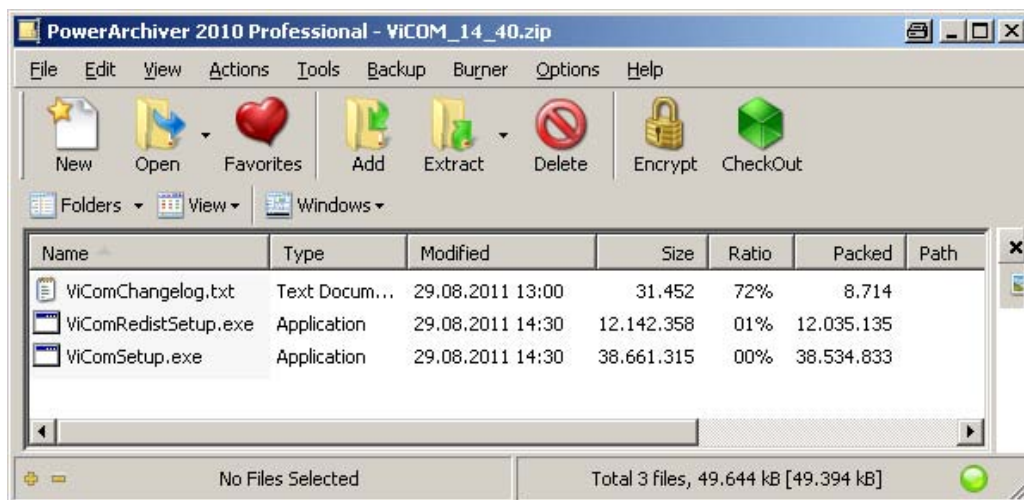
3.1.3 Installing the ViCom Interface Dataset and Demo Applications

Run the ViCom Setup.exe file located on the following subdirectory of the accompanying CD-ROM:

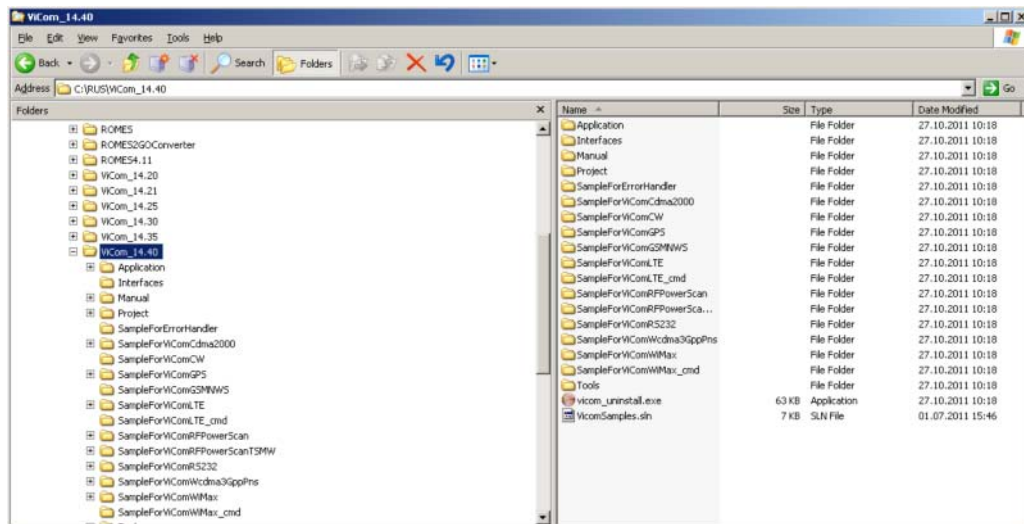
..\ViCom\Vxx_yy\



The R&S ViCom dataset is also available via the R&S homepage. Download and unpack the zip file in a local directory and start ViComSetup.exe to run the ViCom setup. From version 14.30 also a redistributable package ViComRedistSetup.exe is provided. It can be used to install a runtime environment for ViCom applications.



The setup installs the ViCom interface dataset, available test application(s) and instrument utilities including the OptionKeyInstaller and IEEE1394 device driver installer. If the standard instructions are followed, a RuS directory will be created as follows:



3.1.3.1 ViCom Interface Files

The ViCom API can only be used when a specific set of files is installed on the target machine. In this section, the list is given. For each technology, there are some specific DLLs that are required only for the technology besides a basic set of files.

General Files

These are the files that are necessary to run any of the provided technologies.

Dynamic Libraries:	Supporting Files:	Header Files
AppSpecificRuSLogFileInterface.dll	ELF (Firmware) Files:	ViComBasicErrors.h
CMS_IMD.dll	TSMx_Application.elf	ViComBasicInterface.h
CMSDecCDMA.dll	TSMX_Bootware.elf	ViComBasicInterfaceData.h
CMSDecGPRS.dll	PEC Files:	ViComError.h
FFTW.DLL	TSMUFrames.pec	ViComLoader.h
IMD_Base.dll	TSMWFrames.pec	ViComRawDataProcessor.h
k1394api.dll	Microsoft Libraries:	
libfftw3-3.dll	MFC71.dll	
libfftw3f-3.dll	mfc100.dll	
log4cxx.dll	msvcp71.dll	
ModulePool.dll	msvcp100.dll	
PhylisCrashReport.dll	msvcr71.dll	
PhylisEnvironmentInfoProvider.dll	msvcr100.dll	
PhylisFilterDesigner.dll	Configurarian Files:	
PhylisKernelMessageModule.dll	PhylisKernalRuntimeRouting.txt	
PhylisLowLevelErrorMessageHandler.dll	ProcessIdent.txt	
PhylisTimer.dll		
RSToolbox.dll		
TsmuWorker.dll		
TsmwAdmin.dll		
TSMWWorker.dll		
UmtsL3DeclfcMod.dll		
ViComResourceManager.dll		

ViCom WCDMA interface files

To run the ViCom WCDMA API and/or the BCCH demodulator, the list of common files is completed with the following entries:

Dynamic Libraries	Header Files
UmtsDemodulator.dll	ViComWcdma3GppErrors.h
UmstPnsOption2.dll	ViComWcdma3GppPnsInterface.h
ViComWCDMA3GppPns.dll	ViComWcdma3GppPnsInterfaceData.h
ViComWcdma3GppPns.dll	

ViCom GSM Network Scanner related Files

Dynamic Libraries	Header Files
GS3Worker.dll	ViComGSMNWSErrors.h
GsmDemodulator.dll	ViComGSMNWSInterface.h
ViComGSMNWS.dll	ViComGSMNWSInterfaceData.h
ViComGSMNWSw.dll	

ViCom CDMA 2000 Files

Dynamic Libraries	Header Files
Cdma2000Pns.dll	ViComCdma2000Errors.h
C2kDemodulator.dll	ViComCdma2000Interface.h
ViComCdma2000.dll	ViComCdma2000InterfaceData.h
ViComCdma2000w.dll	
EvdoDemodulator.dll	
EvdoPns.dll	

ViCom RF Power Scan Interface Files

Dynamic Libraries	Header Files
SpectrumWorker.dll	ViComRFPowerScanErrors.h
ViComRFPowerScan.dll	ViComRFPowerScanInterface.h
TsmwSpectrum.dll	ViComRFPowerScanInterfaceData.h

ViCom LTE Scanner Files

Dynamic Libraries	Header Files
LteDemodulator.dll	ViComLteErrors.h
LteScanner.dll	ViComLteInterface.h
ViComLtew.dll	ViComLteInterfaceData.h
LteMimoScanner.dll	

ViCom WiMAX Files

Dynamic Libraries	Header Files
WiMaxDemodulator.dll	ViComWiMaxErrors.h
WiMaxScanner.dll	ViComWiMaxInterface.h
WiMaxScannerSync.dll	ViComWiMaxInterfaceData.h
ViComWiMAXw.dll	

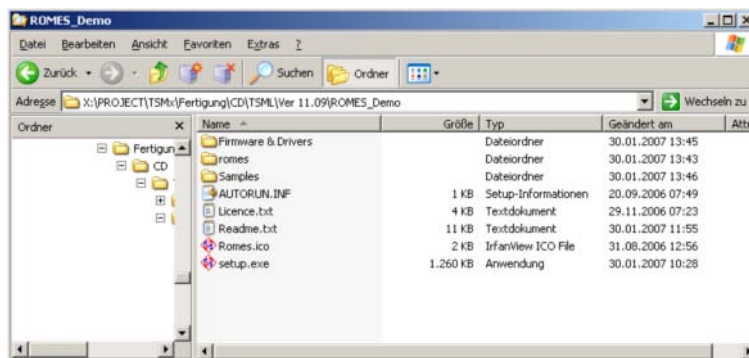
ViCom RS232 Files

Dynamic Libraries	Header Files
ViComRS232.dll	ViComRs232Errors.h
	ViComRs232Interface.h
	ViComRs232InterfaceData.h

3.2 Installing the ROMES Demo Application

The ROMES demo application can be found in the “ROMES demo” directory on the CD-ROM. It is only available for TSMx/TSMW receiver types.

The Setup.exe is located under the “ROMES_demo” directory, as shown below:



Double click on Setup.exe to begin the installation process, and follow the standard installation instructions given by the program.

NOTICE

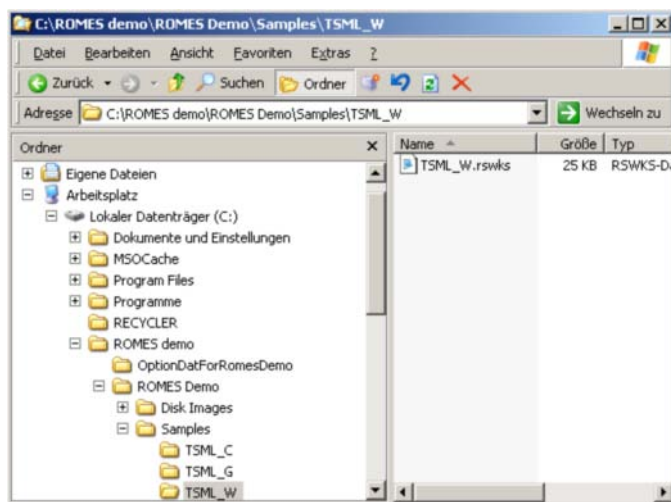
Installing the ROMES demo application requires the powered TSML-x receiver to be connected with the host PC.

3.3 Starting the ROMES Demo Application

Please see the ROMES manual for instructions on how to use the application to control the R&S TSMx.

Sample worksheets are supplied with the ROMES demo program as .rswks files, and are found in the Samples subdirectory of ROMES demo. Under Samples, please see the sub-directory corresponding to your R&S TSMx (as shown below).

Prerequisites if ROMES Demo Application has not been Installed



3.4 Prerequisites if ROMES Demo Application has not been Installed

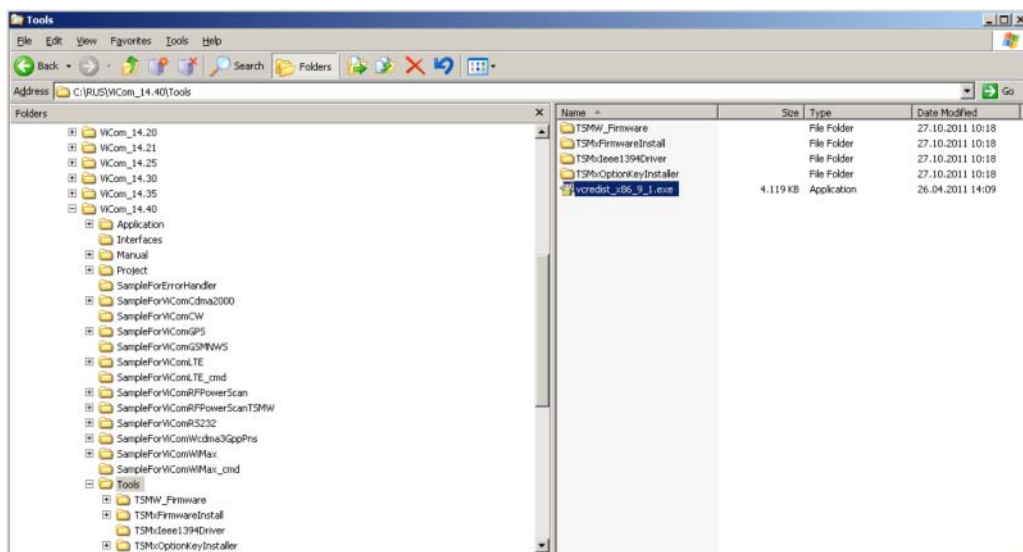
NOTICE

This section is only applicable for TSMx users and users who haven't installed the ROMES demo application

Firstly, it will be necessary to install the Windows device driver for your R&S TSMx manually, if such a device is used. The IEEE1394 install utility has been installed along with the ViCom interface dataset. Please refer to appendix A of this manual for details.

The TSMW does not need any special driver to be installed. However, the PC must have a LAN interface to connect the TSMW to, or the TSMW is connected using some kind of additional LAN hardware like a switch or a hub. It is important that the TSMW can be reached from the PC where ViCom is actually installed.

Secondly, if your R&S TSML-W includes the BCH demodulation option, then the Microsoft libraries MSVCR80.dll and MFC80.dll must be installed. Do this, by double clicking on the file vcredistx86.exe, found in the following location:



You can check to see if your R&S TSMx-W supports BCH demodulation, using the OptionKeyInstaller utility. First install the Windows device driver as described above, and then open the OptionKeyInstaller. Instructions for how to display the available options on the R&S TSMx-W are given in Section 6 of this document. In the “Available Options” tab of the OptionKeyInstaller, you should see the following line:

R&S TSMx-K34 – Option for R&S TSMx OEM: WCDMA BCH Demodulator

3.5 Diagnostics Information

3.5.1 RS232 output with the TSMx

Diagnostic information is available in a text file from the R&S TSMx scanner via the RS232 interface.

Connect the null modem cable into the back of the R&S TSMx, and connect the other end to the appropriate connector on the control computer. Settings for the connection are Baud rate 115200, 8 bit, no parity, 1 stop bit (8-N-1).

A stream of log information comes automatically when the R&S TSMx is running. It can be read with a third party application like Hyperterminal, or saved in a log file by your application.

3.5.2 Managing more than one R&S TSMx scanner

Up to 3 R&S TSMx scanners may be connected in series, via the 2 IEEE 1394 connectors on the rear panel, and used to take simultaneous measurements.

For example, one unit could be used as a GSM network scanner, one as a PN scanner and one as a CW receiver. The position of a R&S TSMx unit in the cascade is irrelevant for its use, just check whether the unit is equipped with the required options.

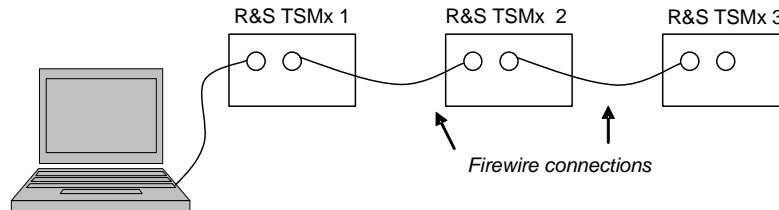


Figure 2 Many TSMx devices connected to one control computer

Short interruptions of the IEEE1394 connection are tolerated. However, if new R&S TSMx devices are connected to the firewire or if a R&S TSMx device is switched off then the ViCom interface must be re-loaded. While the interface is loading, connected R&S TSMx devices must not be switched off, disconnected or reordered.

The ViCom Basic interface function `SelectReceiver` allows an application to select which R&S TSMx scanner to communicate with. For example,

```
bool success =  
  
    pViComIf->GetBasicInterface().SelectReceiver(m_ViComError, 2);
```

would select the third scanner from the computer, R&S TSMx 3 in the above diagram.

4 Programming with the ViCom Interface

This chapter introduces to the ViCom application programming interface. In the later part of this chapter, the basic structure that is common to all different technologies is explained. Beforehand, the basic principle of how measurement tasks are handled on a TSMx device is described to create a basic understanding for some technically enforced decisions.

4.1 Device Details

4.1.1 Measurement Scheduling on the TSMx

The TSMx series has one built-in receiver that can be used to perform different measurement tasks. The measurement tasks cover a variety of technologies and have different requirements on what shall be measured, so the access on the receiver unit has to be managed properly to use it efficiently.

In principle, there are three different groups of tasks the TSMx has to coordinate:

- Periodic measurements, like GSM or WCDMA scans
- Demodulation measurements, like for the BCH demodulator
- Smaller management tasks, like attenuation adaption etc.

Since one TSMx unit can be used to perform several measurement tasks in parallel and all these tasks have different timing constraints, all these tasks have to be prioritized.

Periodic measurements have a configurable measurement rate that the scheduler tries to satisfy. Until no demodulation task has to be performed, the periodic measurements are done, until their measurement rates have been satisfied.

A demodulation request overrides a periodic measurement if one is active. Most demodulations have to be done at a certain time, so it is necessary to perform the decoding then. If nothing is left to do, the receiver simply turns into idle mode until the next job is ready.

Below is a sample configuration, where GSM measurements have to be done at 3 Hz, a WCDMA measurement at 1 Hz and 2 Power Scans in a second. Additionally, a GSM demodulation is requested during the measurement phase.

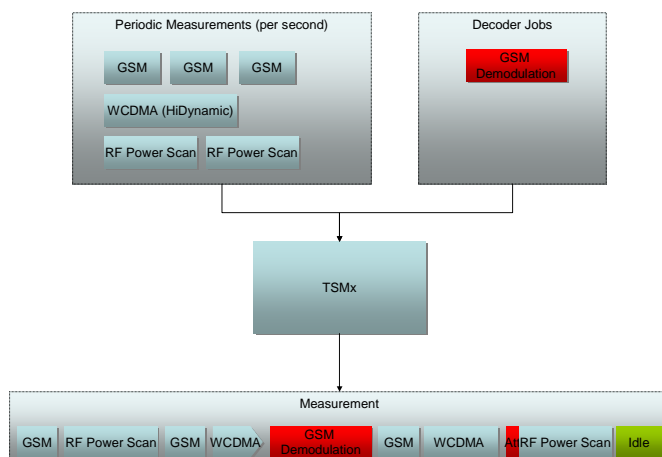


Figure 3 Scheduler Sample

The scheduler would then perform the tasks in a sequence similar to the one depicted above. Since 3 GSM measurements have to be done, the first one is started. Next job is then to perform a RF Power Scan, before a second GSM measurement has its turn. The following WCDMA measurement is stopped since a GSM demodulation request has arrived and must be done immediately. The paused WCDMA measurement is then resumed once the demodulation request has been satisfied.

In the sample, the second RF Power Scan started with an ADC overflow, which causes the attenuation to be modified and the measurement to be restarted. This adds some extra time overhead for the justification and therefore takes this time from the idle pool.



Note that the block sizes have no relation to the actual execution times in the picture above. The TSMx is capable of doing more measurements than shown in this example, but for simplicity the sample has been chosen as described. In principle, more measurements are possible than the number depicted in the figure above.

4.1.2 Resource Allocation on the TSMW

The TSMW has not the same automatic scheduling algorithm as the TSMx devices, but it offers a finer control over the measurement behavior. All measurement tasks that shall be done can be distributed among the two built-in front-ends of the TSMW. The amount of time that each task gets assigned depends on a manually defined maximum load for that task.

For example, if you want the TSMW to do GSM, WCDMA and LTE measurements and demodulation for the former two, it is possible to dedicate the first front-end to the GSM measurement task by 40%, and use the remaining 60% to measure and demodulate WCDMA signals. The second front-end can then be used to demodulate GSM signals and to perform the LTE measurements. This sample is also shown in the figure below.

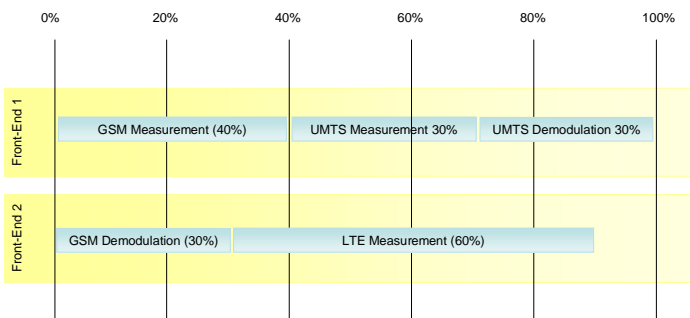


Figure 4 Resource Allocation on the TSMW

As described in the figure, it is possible to not completely utilize all available resources on a front-end. In such a case, the TSMW itself detects that there are still resources left to be used and will then distribute the resources among the measurement tasks.

During the measurement the TSMW will then try to fulfill those constraints as good as possible over a longer period of time. I.e. in the sample above it would try to measure GSM signals 4 seconds in a 10 seconds block, but it is not guaranteed in what order the measurements will be done.

4.1.2.1 Demodulation Constraints

Besides the maximum load defined in the TSMW resources described above, there is an additional constraint that can be specified in the ViCom interfaces concerning the maximum processing time dedicated for BCH demodulation.

This is a harder constraint than the resource allocation scheme described above. It will supersede the resource allocation in any case. For example, if the maximum load in the ViCom is set to 20% and one front-end on the TSMW is allocated for demodulation to 100%, the ViCom will control that no more time is dedicated to the demodulation task than the 20% configured in the interface.

If you use multiple ViCom Interfaces or multiple instances of one ViCom Interface, you have to consider the following:

The load for demodulation specifies a percentage of the full CPU load one ViCom interface instance utilizes on the PC. The maximum value of the load for demodulation is 50%. If multiple ViCom instances (from the same or from different ViCom interfaces) are running in parallel, the Operating System of the PC handles the CPU allocation for each ViCom instance.

E.g.:

ViCom instance #1 uses 40% of the PC's CPU load

ViCom instance #2 uses 60% of the PC's CPU load

Demodulator load of ViCom #1 is set to 50%

Demodulator load of ViCom #2 is set to 40%

This leads to:

Total CPU load for Demodulation of ViCom instance #1 is $40\% \cdot 50\% = 20\%$

Total CPU load for Demodulation of ViCom instance #2 is $60\% \cdot 40\% = 24\%$

4.2 Start Programming

This section describes what a developer needs to do to control one R&S TSMx/TSMW scanner, unless otherwise mentioned. All the functions and data structures mentioned in this section are also described in detail in the online reference manual that comes with the ViCom delivery.

ViCom interfaces include the following groups:

- Error functions, common to every ViCom interface
- Loader functions, common to every ViCom interface
- Basic functions, common to every ViCom interface
- Specific functions that belong to a particular ViCom interface, e.g. the functions of the class `ViComWcdma3GppPnsInterface`

The error and loader functions are used to manage the interface. See the section below for an explanation of how to use these functions.

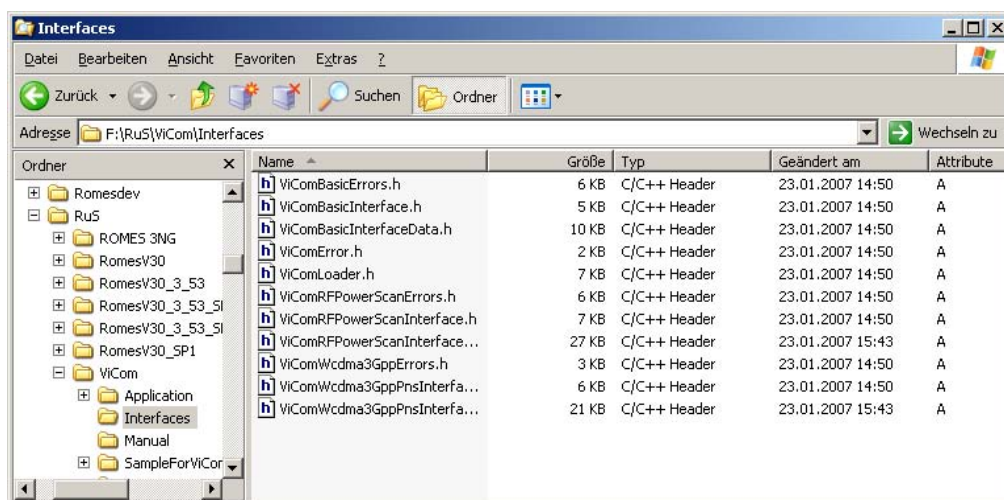
The basic and specific interface functions carry out the actual tasks of setting parameters on the scanner and retrieving measurements. Note that where a function returns a data structure, this data is valid as long as no other interface function is called, either to change settings or retrieve new information.

There are no callbacks on the ViCom interface, but R&S TSMx measurements may be retrieved regularly from the ViCom buffer. The requesting function is able to sleep for up to a specified timeout period, in case no results are available in the buffer. When measurements are being made and read regularly, this has the effect of synchronising the measurement requests to the incoming data.

The calling thread will be blocked while ViCom waits for the measurement function to return, but CPU resources will not be used up by repeated function calls or looping.

4.2.1 Prerequisites

- The Firewire connection to the R&S TSMx receiver to be successfully installed as described in Section 2 “Getting Started” above, or the TSMW can be reached via LAN
- Visual Studio 2003, Version 7.1.3088 or later
- ViCom Interface header files that you will need to include in your project are found in the sub-directory named Interfaces.
- R&S ViCom interface libraries listed in Section 2 above. The interface libraries and other files are found in the sub-directory named Application.
- Microsoft libraries listed in Section 2 above, also supplied in the same zip file, in the Application sub-directory.



4.2.2 On Startup

A typical series of steps is described below, that an application might follow in order to start using an R&S TSMx/TSMW scanner.

Create a `CViComError` object, to be used as a parameter in ViCom functions, to get an error code if the function fails.

4.2.2.1 Connecting to the TSMx

Create a `CViComLoader` object to get the ViCom interface that you need. To do this, you must instantiate the `CViComLoader` template with the chosen interface class, for example:

```
CViComLoader<CViComWcdma3GppPnsInterface> myViComLoader;
```

After the application has finished initialising, call the `CViComLoader` function, `LoadTsmx()`, with the `CViComError` object as a parameter, for example:

```
bool bLoaded = myViComLoader.LoadTsmx(myViComError);
```

Loading the chosen interface will take 20-30 seconds, since the complete firewire chain is searched for all connected devices.

4.2.2.2 Connecting to the TSMW

The handling is somewhat different when using the TSMW device. The aforementioned loader object has to be created with an additional template parameter. This parameter is also used in the TSMx sample, but there is an appropriate default to keep existing code working.

Writing a line as shown below will create a ViCom loader object that can be used to connect interface with a specific TSMW device.

```
CViComLoader<CViComLteInterface, CViComLoader_TSMW>
myTsmwViComLoader,
```

To actually start communication between the PC and the TSMW device, use the `ConnectTsmw()` function made available in the new ViCom loader object:

```
bool bLoaded = myTsmwViComLoader.ConnectTsmw ("192.168.0.2");
```

4.2.2.3 Accessing the ViCom Interface

If at least one device has been loaded successfully, then `GetpInterface()` can be called, to get a pointer to the function interface, for example:

```
CViComWcdma3GppPnsInterface *myIF =
    myViComLoader.GetpInterface(myViComError);
```

The chosen ViCom interface can now be used, via its pointer. Interface functions that are specific to the instantiated interface are accessed directly, for example:

```
myViComIF->SetFrequencyTable(myViComError, mySettings->ChannelSettings)
```

Interface functions that are common to all ViCom interfaces, and are part of the Basic Interface set, are accessed via the `GetBasicInterface()` function. For example (when working with a TSMx device):

```
bool success =
myViComIF->GetBasicInterface().SelectReceiver(myViComError,
                                             mySettings->dwReceiverIndex)
```

The ViCom Reference manual lists the functions that belongs to a single technology interface and those that belong to the BasicInterface.

4.2.3 R&S TSMx Management Functions

`GetConnectedReceivers()` is a basic interface function, available in all ViCom interfaces, that returns details of all the R&S TSMx scanners connected to the control computer, in the structure `SConnectedReceiverTable`. For example:

```
SConnectedReceiverTable myReceivers =
    myViComIF->GetBasicInterface().GetConnectedReceivers(myViComError);
```

where `myViComIF` is a pointer to a ViCom object. Note that `GetConnectedReceivers` is accessed via the `GetBasicInterface` function.

When a function that takes a `ViComError` reference as a parameter returns successfully, the `ViComError` code will be 0.

If an error is returned, the programmer can read the error code from `myViComError` using the function `GetErrorCode` as follows:

```
DWORD myErrorCode = myViComError.GetErrorCode();
```

A standard ViCom error message with more information can be read from `myViComError` by calling `GetErrorString`:

```
string myErrorString = myViComError.GetErrorString();
```

4.2.4 Using the ViCom data structures

It is useful to create and use your own ViCom data structures, for example during processing ViCom data that has been gathered using the interface functions. Programmers can access the ViCom data structures by creating a class that inherits from the interface data class, `CViComWcdma3GppPnsInterfaceData`.

For example:

```
class MySettingsClass : public CObject, public
                      CViComWcdma3GppPnsInterfaceData::SSettings
```

In this example, the programmer has created a class that inherits `SSettings` from `CViComWcdma3GppPnsInterfaceData`.

4.2.5 Reading and Changing Settings

The current settings of the selected R&S TSMx/TSMW scanner can be read, before or during measuring, using the following function:

```
GetSettings()
```

The current settings of the selected R&S TSMx/TSMW scanner can be changed using the following functions:

- `SetFrequencyTable()`
- `SetMeasurementMode()`
- `SetPduRequests()`
- `SetResultBufferDepth()`
- `SetTimebaseSynchronisationMode()`

More details of these functions can be found in the ViCom reference manual.

4.2.6 Getting Measurements

Note that `SetFrequencyTable()` should be called before measurement is started! An application must specify which frequencies the R&S TSMx should scan before starting to measure. The R&S TSMx/TSMW has no default frequency values.

Start taking measurements by calling the `StartMeasurement()` method on the basic interface, for example:

```
myViComIF->GetBasicInterface().StartMeasurement(myViComError;
```



Note that `StartMeasurement` is a member of the Basic part of the ViCom interface, therefore it must be called via the `GetBasicInterface()` function.

It is useful to get information from the R&S TSMx while it is preparing to take measurements. Do this by calling the function

```
GetTSMxMessagesDuringStartMeasurement()
```

soon after calling `StartMeasurement()`.

`GetTSMxMessagesDuringStartMeasurement()` returns a list of messages, which may be Information, Warnings or Errors.


```
SViComList<CViComBasicInterface::SMessage> *myMessageList =
pViComIf->GetBasicInterface().
    GetTSMxMessagesDuringStartMeasurement(myViComError);
```

After `StartTakingMeasurements` has returned successfully, measurements can be retrieved from ViCom's result buffer. It may be useful to know, how many messages are in the buffer, whether it has overflowed and if so, how many measurements have been lost. Do this by calling the ViCom Basic interface function, `GetResultCounters()`:

```
CViComBasicInterface::SResultCounters *myResultCounters =
myViComIF->GetBasicInterface().GetResultCounters(myViComError);
```

The measurements can be retrieved from the result buffer by repeatedly calling the ViCom interface function `GetResult()`, which returns a single measurement result in the `SMeasResult` structure. For example, to read a hundred measurements:

```
for (DWORD i = 0; i < 100; i++)
{
    CViComWcdma3GppPnsInterface::SMeasResult *myResult =
        myViComIF->GetResult(myViComError, myTimeoutInMs);

    //
    // Do something with the results here...
    //
}
```

`myTimeoutInMs` is the time in milliseconds that the function should wait if no results are available.

To stop taking measurements, call the ViCom Basic interface function,

```
StopMeasurement().
myViComIF->GetBasicInterface().StopMeasurement(myViComError);
```

To check that the R&S TSMx/TSMW has stopped measuring, call the ViCom Basic interface function `HasMeasurementStopped()`. In the following example, `HasMeasurementStopped()` is called with a timeout of 5 seconds. The application thus checks to see if measurement stops within 5 seconds.

```
bool *pbStopped =
    myViComIF->GetBasicInterface().HasMeasurementStopped(myViComError, 5000);
```

4.2.7 Releasing the Scanner

The ViCom loader function `ReleaseTsmx` releases all resources used by ViCom, and resets the R&S TSMx.

```
myViComLoader.ReleaseTsmx(myViComError);
or, when using a TSMW:
myTsmwViComLoader.DisconnectTsmw();
```

4.3 Differences between TSMx and TSMW

The main interface functions can be used for all TSMx/TSMW devices supported by ViCom. However, there are few but subtle differences that are revealed here.

4.3.1.1 Selecting the Device

The TSMW is loaded using its unique IP address (unique at least in the local network), whereas the TSMx models are identified by their position in the firewire chain. In the latter case the `SetReceiverIndex()` function can be used to load TSMx devices. When working with the TSMW, this function should not be used. It will return an error code (129) if called in this case, indicating that calling that function does not change anything. This also applies to the `GetConnectedReceivers()` method, which only makes sense in the context of the TSMx.

The TSMW on the other hand is not connected via FireWire, but uses a GigaBit LAN connection to the host PC. Therefore, there is no receiver index in that case, but a single IP address that the device has (see appendix TSMW Configuration).

This IP address must be specified when creating the `CViComLoader<ViComInterfaceName, CViComLoader_TSMW>` instance used to create the actual technology specific ViCom interface.

4.3.1.2 Resource Acquisition

The TSMW enhances some of the interface functions to be able to select the front-end used for the configured measurement, since the device itself has two front-ends built into it. The different ViCom interfaces then provide was to distribute a measurement on one of those front-ends, also defining how many percent of the time the front-end shall be used for that task.

The combination of front-end identifier and load is called a TSMW resource, and such resources can be obtained by asking the `CViComBasicInterface::GetTSMWResource()` for a handle. These handles must then be set in the technology specific measurement configuration when using the TSMW. Requesting a specific amount of time on a front-end may also result in an error code returned, since the front-end might already have too many resource handles given away to serve the recent request.

When the measurement is stopped and the ViCom interface is released, the allocated resource handle must be returned to the stock of resources again to make it possible for other measurements to reuse that data again.

This mechanism is also reflected in the different sample applications that come with the ViCom interfaces. Each sample application that can load TSMx and TSMW devices offers a receiver configuration similar to the one shown in the figure below. When selecting a TSMW receiver, the IP address can be entered in the Address field and the measurement and demodulator front-end usage can be set.

The image shows a software dialog box titled "Load PN-Scanner". It is divided into three main sections:

- Receiver Selection:** Contains a "Type" dropdown menu set to "TSMW" and an "Address" text field containing "192.168.0.2".
- Scanner Settings:** Contains a "TSMW Frontend" dropdown menu set to "FE-1" and a "TSMW Load [%]" text field set to "90".
- Demodulator Settings:** Contains a "TSMW Frontend" dropdown menu set to "FE-1" and a "TSMW Load [%]" text field set to "10".

Figure 5 TSMW Settings

4.4 Using the Demodulators

Many of the technologies supported by the scanners can additionally demodulate information sent on specific channels of the underlying RAT. It is even possible to receive a textual representation of the information in most cases (as long as the internal decoder supports the appropriate standard).

In each chapter, special constraints of the specific implementations are described, besides the usage of the sample application and a coding example. Refer to the appropriate chapter to learn more about the implementation.

4.4.1.1 General Concept

The API provides two ways to configure the demodulation process, depending on the current state of the TSMx. When there is no measurement active at the moment, the initial configuration is made. In that phase many parameters of the demodulation process can be specified. During the measurement different commands can be used to change the demodulation. For example, internal results can be reset to enforce another demodulation of a message already decoded.

The demodulator is configured by specifying a set of data records. Each data record is either used to define how a specific message type (PDU, System Type Information etc.) is demodulated on a specific channel. Several modes are available to do the actual demodulation, some performing their job automatically, some requiring additional user input. Some of the so-called demodulation modes require a timeout value as additional parameter.

In summary there are up to five arguments that have to be specified:

- Channel Index / Frequency Index
- Message Type (PDU, System Type etc.)
- Demodulation Mode (see below)
- Timeout (only required for special modes)
- BTS reference / Node-B reference (only required for special modes during measurement).

4.4.1.2 Demodulation Modes

The demodulator API allows you to specify in detail when and how often a message type shall be decoded. The available options depend on whether the decoding is defined at start of the measurement, or if a single demodulation request is issued during the measurement.

Start of Measurement

When a measurement is started, for each channel and message type the following decoding modes can be set:

- Demodulate Once (0): Each message type is only decoded one time for each basestation. After a decoding result is available, the message type is never updated again. Fastest solution, but can lead into troubles (in rare cases) when using on long drive tests and the same basestation indices are assigned to different basestations.

- Repeated Demodulation (2): Decode a message as fast as possible, and then again after N seconds. In the API, the delay is specified in 100 ms, so a value of 10 means a delay of approx. 1 second. The delay specifies the time the demodulator waits after a complete demodulation of the SIB has been finished before a new one is started. As a consequence, the new result will be available after the delay plus the time needed to perform a new demodulation.

For an example, refer to the figure below. A repetitive demodulation of SIB 7 (in the case of WCDMA) is demonstrated. After the SIB7 has been demodulated completely, the TSMx does not attempt to do another demodulation until the delay time has elapsed. Afterwards the next demodulation attempt is started. The same applies to other technologies as well.

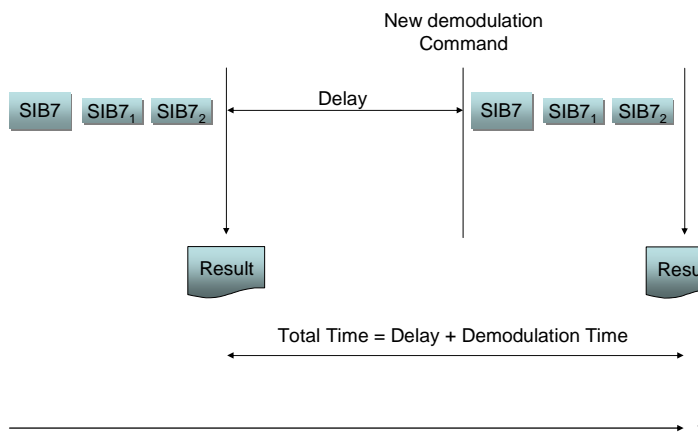


Figure 6 Automatic Repetitive Demodulation

- Demodulate on Command (1): The message is not decoded automatically. A special decoding command has to be sent during the measurement to start decoding, which must then specify for which Node B the demodulation shall be done.

It is important that for each channel and message type combination that has to be decoded at some time during the measurement, one of those three settings is specified before the measurement is started. For messages that shall not be decoded all the time but only in special cases, the “Demodulate on Command” setting must be chosen.

All those settings apply to all basestations found on one channel. Basestation specific decoding is only possible during measurement (see below).



The numbers in braces after the demodulation type refer to the number that is assigned to the mode in the API. Since they are part of an enumeration, names similar to the appropriate meaning are also available, so you normally want to deal with those identifiers.

During the Measurement

In some cases it might be desired to save resources and decode messages only for special basestations or under certain circumstances. As mentioned above, for such messages the “Decode on Command” setting has to be specified before a measurement is started. At any point when a measurement is active, it is then possible to issue one of the decoding commands below:

- **Demodulate BTS (3):** For a special basestation / Node-B on one channel, this command tries to decode a defined message. To avoid too much resource allocation, a time-out can be specified when using this command. If the specified time is consumed and no decoding could be done, no result is returned and the resources are freed for other tasks. The timeout can be 0 as well, in which case the demodulation is performed until it succeeds.
- **Demodulate BTS High Power (4):** Similar to the mode described above, but utilizes more receiver resources, since no probability calculation is done to reduce resource allocation like above. The TSMx then does all it can do demodulate the result in the given time constraint (if specified).
As for the “Demodulate BTS”, a time-out can be set for this operation in which it must complete. Otherwise, the request is stopped. The time-out can again be 0, which tells the TSMx to try demodulation until it succeeds.

Both commands have a second variation. The difference is that in both cases that old results might be returned if available. These commands are: “Demodulate BTS (Old)” (7) and “Demodulate BTS High Power (Old)” (8). The old content can be deleted with the Reset command described below.

It is also possible to clear the contents of the internal message containers associated with a channel and the BTS. Two commands are designed for that purpose:

- **Reset BTS Demodulation Cache (5):** Clears only the frame container content within one basestation of a channel. This can be used to explicitly restart demodulation for a specific cell.
- **Reset Channel Demodulation Cache (6):** This clear all demodulation results associated with a channel. After this command, the demodulation for that channel will restart according to the configuration made before the measurement was started.

These commands can be issued only during the measurement, of course. Compared to the commands mentioned earlier, these will have an effect immediately since they don't depend on the availability of a signaling. There is no “Undo” functionality for this, so use these carefully if you work on a highly efficient TSMx/TSMW project.

4.5 Setting up a Custom Project

This section describes how to set up your own project that enables you to work with the ViCom API and that helps you to create a distributable application. Perform the tasks described below step-by-step and you should end up with a ready to start application frame.

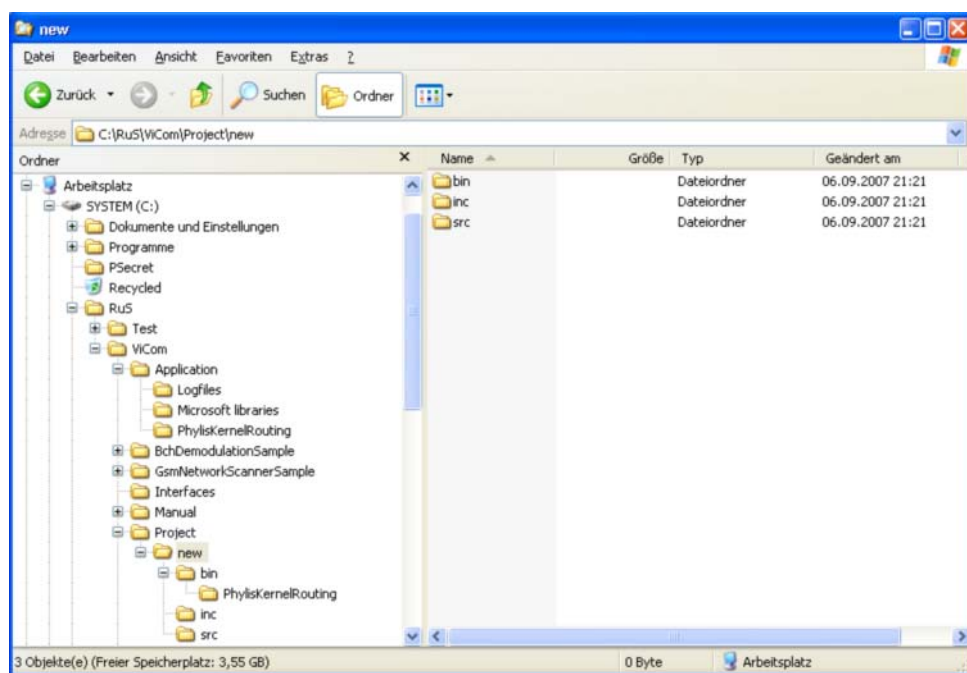


In the subsequent chapters (small) demo applications are shown as code pieces. These can simply be put into the main C++ file.

4.5.1 Project Structure

We recommend a standard project structure for your own developments that consists of three folders. One folder holds the source data (it is called “src”), one the header file provided by the API in the “inc” folder and the last folder holds the application environment (“bin”). The content of the latter folder can be used as base for your own program installer once you finished the development of your application.

This configuration is depicted in the figure below.



The content of the folder can be moved and copied to any location on your computer and should still work. It is not necessary to keep it where it has been created.

4.5.2 Create the Project Environment

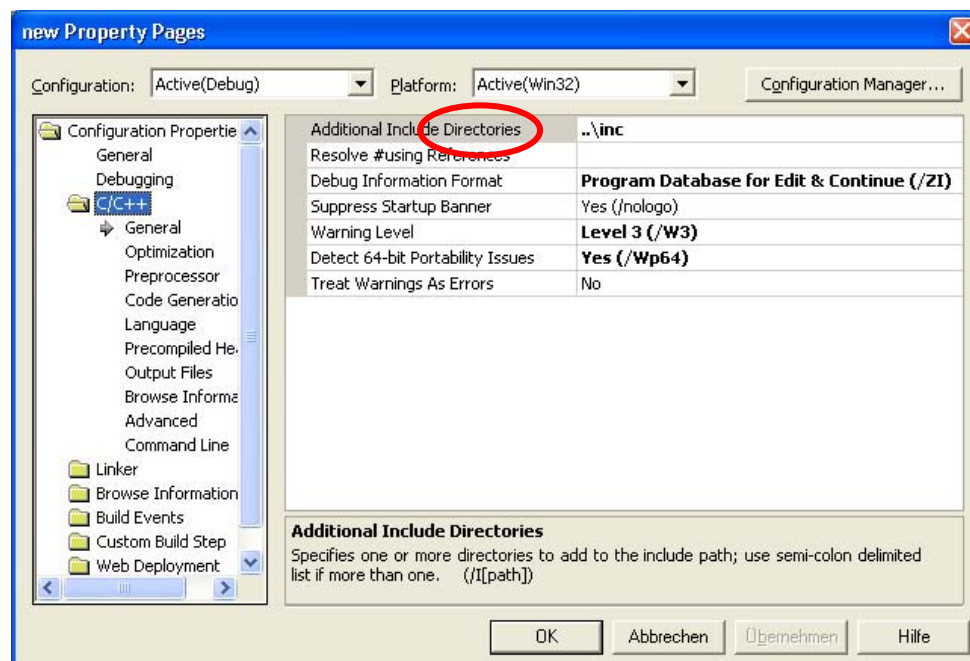
To compile your demo project you need the ViCom header files, to run the application you need a set of DLLs (and other files). In the “Project” folder of the ViCom installation directory you can find a script called “createProject.cmd”. Run this script, providing it with a command line argument that holds the name of the project folder to create. For example, run the following command line to get the same result as shown in the screenshot above:

```
C:\RuS\ViCom\Project>createProject new
```

4.5.3 Project Settings

In the source folder you will find a Visual Studio .NET 2003 ® project that is set up to work in the created folder structure. The project is a Windows console project, so if you need some other type of project (like one for a MFC application), you should carefully read through the following paragraphs. There you can find the important settings that make the project run. When the console project is sufficient for your needs, you can skip the rest of the section.

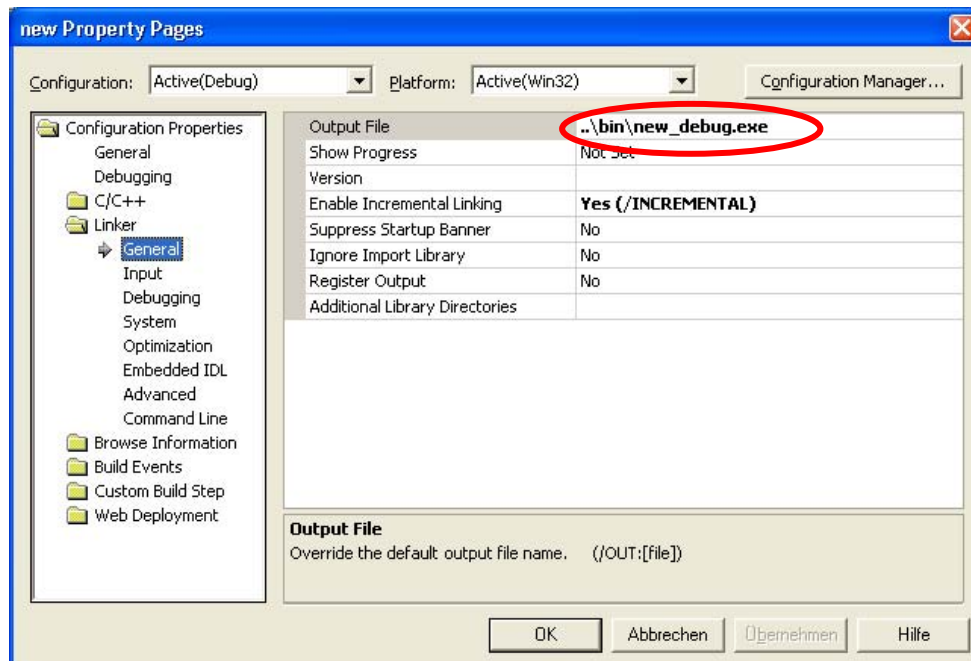
The first important part of the project configuration is the default inclusion of the ViCom headers. The header files were copied while running the script to the “inc” folder, which therefore must set in the “C\C++\General” settings.



The second important change in the default project settings is the output file. The application executable is required to run a specific environment to use the ViCom functionality. The phylis kernel and its configuration as the central element in the ViCom libraries must be found in the same directory resp. in the directory “PhylisKernelRouting” folder.

Therefore, the output file is set in the sample project to be placed into the “bin” folder. In the property pages, this can be specified in the section “Linker\General” as shown in the screenshots below.

One important thing to know is that the ViCom API does not require any additional linkage specifications. The complete API is made of dynamically loaded DLLs and the interface header files, so no settings have to be changed in the “Linker\Input” tab.



4.5.4 Working with the Code

The first steps when setting up a ViCom application is to select which technology you want to use (depending on your scanner, only some parts of the ViCom functionality can be used), and include the related header files.

In the automatically generated main.cpp file, you find a minimal ViCom sample program that you can use as starting point for a simple console application. The code is shown below:

```
#include "stdafx.h"

#include <windows.h>

#include "ViComBasicInterface.h"
#include "ViComError.h"
#include "ViComLoader.h"
#include "ViComRS232Errors.h"
#include "ViComRS232Interface.h"
#include "ViComRS232InterfaceData.h"

/*****

int _tmain(int argc, _TCHAR* argv[])
{
    CViComError cError;
```



```
CViComLoader<CViComRS232Interface> cLoader;  
cLoader.LoadTsmx( cError );  
  
CViComRS232Interface* pInterface = cLoader.GetpInterface( cError );  
  
// TODO: specify measurement settings  
  
// TODO: start measurement  
  
// TODO: stop measurement  
  
cLoader.ReleaseTsmx( cError );  
  
return 0;  
}
```

Important things to notice are the header files that are included at the very beginning of the file. The `<windows.h>` file is required for the definition of the `DWORD` datatype. The include statement for the `CViComBasicInterface`, the `CViComError` and `CViComLoader` classes are necessary no matter what kind of ViCom application you want to write. The header files for the RS232 ViCom API are just included to make the example more meaningful. This part must be adapted according to the technology that shall be used in your application.

The very basic code structure in the main procedure is similar for all ViCom application as well. In any case, you need a `CViComError` object that you can pass in any call to a ViCom method as first input parameter. The next step normally is to load the TSMx device and get a pointer to the interface object that is used to control the device. Further information can be found in the chapters that cover the special technologies and in the introduction above (Start Programming on page 27).

4.5.5 Installation Issues

Once you finished your application and you want to deploy it to your customers, the application result must be packaged into some installation format. The ViCom does not require any special operating system settings to be made, the content of the “bin” directory can directly be used for deployment.



Make sure that you don't forget to remove the debug version of your program from the bin folder before shipping the content of the bin folder.

Nevertheless, some additional steps must be made to properly setup a customer's computer to run ViCom applications on it:

- The Firewire driver must be installed on the target machine to connect to a TSMx device. It must be performed before the ViCom application is used.
- The libraries (dlls) under “%VICOM_HOME%\Application\Microsoft Libraries” must either be put into the windows system 32 directory or into the applications home dir (where your .exe program is executed).

- The redistributable installer package (can also in the directory “%VICOM_HOME%\Application\Microsoft Libraries”) must be installed. Therefore only the self-extracting exe must be called.

4.5.6 Updating the projects

Once a project has been created with the procedure described above, the files are different than the ones in the main ViCom directory. To keep them up-to-date when new releases are installed, a new sample script is added to the ViCom delivery. It is used similar to the createProject script and takes the project name as the only input argument. So, the command

```
C:\RuS\ViCom\Project>updateProject new
```

would copy the most-recent files from the top-level installation directory into the project path. This includes the interface headers, binaries and other stuff.

4.6 Debugging and Error Handling

4.6.1 Debugging Techniques

When debugging an application that utilizes the ViCom API to access one or more devices from the TSMx series, some registry keys control the way the underlying framework behaves in terms of logging, error handling etc. These registry settings can be controlled by some registry files that are stored in the ViCom\LogFiles folder of your ViCom installation. This section describes those registry settings in detail.

The LogFiles directory is also used to store the output files of the sample applications. In that directory, the following registry setting files are also available:

- `PhylisKernelTraceSettings.reg`: Used to enable the internal message tracing facility of the dispatching mechanism, the so-called Phylis Kernel. Normally you don't want to use this, but when a technically complicated problem occurs, you might be requested by the Rohde&Schwarz support to enable the tracing and transfer the output to our technical experts.
- `TsmuWorkerDebugSettings.reg`: When the content of this file is inserted into the registry, the debugging capabilities of the ViCom applications are improved. Normally, the TSMx devices loose connection to the driver very quickly when no processing can be done on the PC side. That is the case when you try to debug your application that utilizes the ViCom API and stop at a specific location with a breakpoint. To leverage the problems that occur in that situation, load this file into the registry and you won't get bothered with the “Connection lost” message any longer.
- `SetPhylisLogPath.reg`: Used to adjust the location of the LogFiles folder of the ViCom installation. Per default this registry file changes the location to `C:\Temp\ViCom Phylis Logs`. In order to manually specify another path one has to open the `SetPhylisLogPath.reg` file in a text editor (e.g. notepad) and change the `RuSPhylModDir=C:\\Temp\\ViCom Phylis Logs\\` entry appropriately.
- `RemoveAllViComRegistrySettings.reg`: Used to clear the registry from all previously made settings.

4.6.2 Message Handler

Underneath the surface, the ViCom API utilizes a Rohde&Schwarz internal mechanism, the so-called PhylisKernel. This message dispatching mechanism is mostly used in graphical environments, which makes it use the possibility to get feedback from the user in some way, especially when it comes to error handling.

This may not be the desired behavior when the ViCom API is used. Usage scenarios may contain environments where no user feedback is possible or where software components have to run in an unattended way.

For such cases, the PhylisKernel supports a mechanism to overwrite the default behavior of asking the user for confirmation in several ways:

4.6.2.1 Error Handling Mechanism

The error handling mechanism in the Phylis Kernel is designed to be extensible by providing a new DLL with a specific API. ViCom itself ships with a default Error Handler that supports the configuration of the error handling as described below.

In case that a message box shall be shown, the kernel checks if it can load a user defined DLL (see below). If such a DLL is found and it exports a function named "DumpLowLevelErrorMessage" with the signature shown below, then it calls that function to take care about the error message. The DLL can handle the error message in any way it wants to.

```
extern "C" __declspec( dllexport ) int
DumpLowLevelErrorMessage(LPCSTR lpszText, LPCSTR lpszCaption,
                        UINT nType)
```

To extend the default behavior, you have to put a DLL named UserLowLevelErrorMessageHandler.dll (resp. UserLowLevelErrorMessageHandlerd.dll in Debug Mode, if there is any different reaction required in these two modes; otherwise one of the two is enough) into the Application folder of the ViCom installation directory.

ViCom is shipped with a default handler DLL that supports disabling the message logging using the registry (see below).

You can use the SampleForErrorHandler sample project as base for your own DLL. This project contains a configuration which shows the above mentioned message box in case of a Phylis Kernel error. The ViCom setup package supplies this sample only as source code. Therefore you have to build the project, if you want to use the UserLowLevelErrorMessageHandler.dll.

```
#include "stdafx.h"
#include <afxwin.h>

extern "C" __declspec( dllexport ) int DumpLowLevelErrorMessage(LPCSTR lpszText,
LPCSTR lpszCaption, UINT nType)
{
    //
    // TODO:
    // Integrate your personal error handling routines here.
    //
    CString csMsg;
    csMsg.Format(_T("Module: %s\nMessage: %s\n"), (CString)lpszCaption,
                (CString)lpszText);
    AfxMessageBox( csMsg, MB_OK | MB_ICONINFORMATION );

    return 0;
}
```

4.6.2.2 Registry

The default error handler DLL that is shipped with the ViCom delivery is called `PhylisLowLevelErrorMessageHandler.dll` (or `PhylisLowLevelErrorMessageHandlerd.dll` in debug mode) and can be used to control the behavior using the registry. In the key “HKCU\SOFTWARE\Rohde&Schwarz\PhylisModules”, the two values “DisableLowLevelMessages” and “DisableLowLevelMessageLogFile” can be used to control whether message shall be written to a file and if message boxes shall be displayed.

If the first value is set to a value not equal to 0, the message will not get written to a file named “LowLevelPhylisMessage XXX.txt”, where XXX is replaced by a timestamp. If the second value is specified and set to a value which unequals 0, no message boxes are shown.

5 ViCom WCDMA PN Scans

5.1 Measuring with WCDMA PN Scanner

The TSMx/TSMW can measure basic RF parameters of any Node B that is transmitting within range.

For the Primary and Secondary Synchronisation Channels (P-SCH and S-SCH) this includes:

- Correlation result of all P-SCHs found at the requested frequency
- Relative power of detected peaks in dB
- Time delay of detected peaks in us

For each Primary Common Pilot Channel (P-CPICH) found at the requested frequency, this includes:

- Scrambling Code
- Total Power of the CPICH

For each Scrambling Code:

- CPICH Channel Impulse Response (CIR) measurements
- Received Signal Code Power (RSCP) and Interference Signal Code Power (ISCP)
- Time drift of C-PICH CIR measurements, giving an indication of Node B timing drift.
- Root Mean Square (RMS) of delay spread related to Chip
- Optionally, frequency (Doppler) drift of the Node B signal

For each peak within the CIR measurement of one Scrambling Code:

- Power of the identified peak
- Relative time of arrival

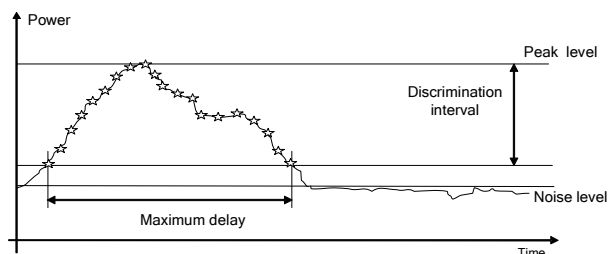
To support the above measurements, (and when this option is present) the TSMx uses the synchronisation channels to synchronise to the Broadcast Channel (BCH) of the same network, and decodes System Information Broadcasts (SIBs). SIB information is available across the ViCom interface, in the measurement result structure. A complete list of information available in each SIB is given in Reference 6, Section 10.

5.1.1 CPICH Channel Impulse Response (CIR) Measurements

The R&S TSMx/TSMW returns a CIR measurement for each Common Pilot Channel (CPICH) that was found during the measurement time, on the requested frequency. This will include the C-PICHs transmitted by different Node-Bs belonging to the same network, in the reception area. Each Node-B broadcasts the CPICH with a different scrambling code, as well as with a different delay to the code broadcasts, because the Node Bs are not synchronised.

A channel impulse response is a measurement of the effect on an impulse or sequence of impulses, of transmitting it along a particular channel. The R&S TSMx/TSMW measures the effect on a sequence of impulses, when a Node B transmits them along the CPICH transmitted by a Node B at the requested frequency.

From each impulse, the R&S TSMx/TSMW measures a power delay profile, by sampling the power at regular time intervals along the pulse (see figure below).



The R&S TSMx/TSMW estimates a lower level for the code power that is high enough above the noise floor to be sure that noise measurements are not included in the CIR measurement. It then gets an array of code power measurements sampled from the signal over the discrimination interval (see below).

The maximum delay is the interval between the first code measurement of the CIR and the last one.

The signal is likely to have reached the receiver via several different reflected paths (multipath). The resulting peaks may overlap, leading to a broader peak, possibly with several sub-peaks. Alternatively, there may be adjacent peaks close together, from the same Node B CPICH transmission.

The signal quality can be estimated by measuring the width of the peak (maximum delay), and the delay to the impulse peak, as well as the code power to inband power ratio E_c / I_0 . The RMS delay spread (S) measures the standard deviation of the delay spread. Delay spread is an indication of the average drift of signals. Delay spread is a common problem, and is very important for optimisation. High delay spreads mean that more fingers are needed in a rake receiver to make a connection between a mobile and a Node B.

For each CPICH signal found, the R&S TSMx/TSMW also returns the Received Signal Code Power (RSCP), and the Interference Signal Code Power (ISCP). The RSCP is the average power of the received signal after despreading and combining. The ISCP is the interference on the received signal in the given timeslot that cannot be eliminated by the receiver.

Together with the spreading factor (SF), these two values can be used to calculate the Signal to Interference ratio (SIR). The following calculation is taken from Reference 8 Section 5.

$$SIR = (RSCP / ISCP) * SF$$

The ratio E_c/I_0 , the average chip energy divided by the total inband energy per chip duration, may be calculated by subtracting the inband power from the absolute code power, both reported in the measurement result structure.

The R&S TSMx/TSMW returns a time drift for each CPICH measurement. It also returns a standard deviation for an approximated Gaussian distribution of time drifts of the PDPs. The accuracy of the time drift can be estimated by considering the standard deviation, ie if it is low, then the calculated time drift is likely to be accurate.

5.1.2 Peak Information

If the signal of one CPICH has reached the scanner via several different paths, there may be several peaks, either overlapping, or single peaks. The R&S TSMx/TSMW separates these peaks and returns a list of structures containing peak information. The information returned includes the time delay of the peak, relative to the start of the measurement, and a measurement of the peak power.

Each time that the R&S TSMx/TSMW synchronises to a CPICH by using the Primary and Secondary Synchronisation Channels, it also returns the inband power and the code power measurements for the PSCH and SSCH. If the R&S TSMx/TSMW is reporting a measurement of a CPICH to which it synchronised previously, then these values will not be reported.

In the peak information structure, the R&S TSMx/TSMW will also report an estimate of the frequency drift of the CPICH signal. This frequency drift could be caused by movement of the receiver (if it is mounted in a moving vehicle for example). However, if the receiver is stationary, then the frequency drift of a Node B can be measured.

5.2 Sample Application

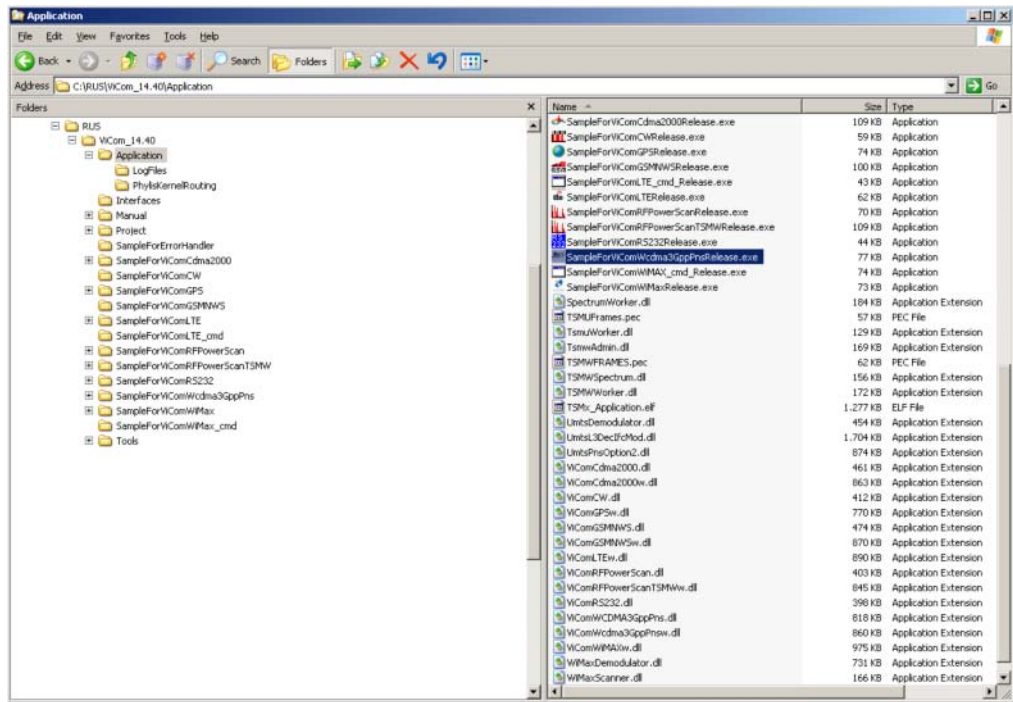
The sample application is a simple implementation of the ViCom interface, to control one PN-Scanner. It may be used

- To test the R&S R&S TSMx/TSMW scanner
- To understand the ViCom programming interface

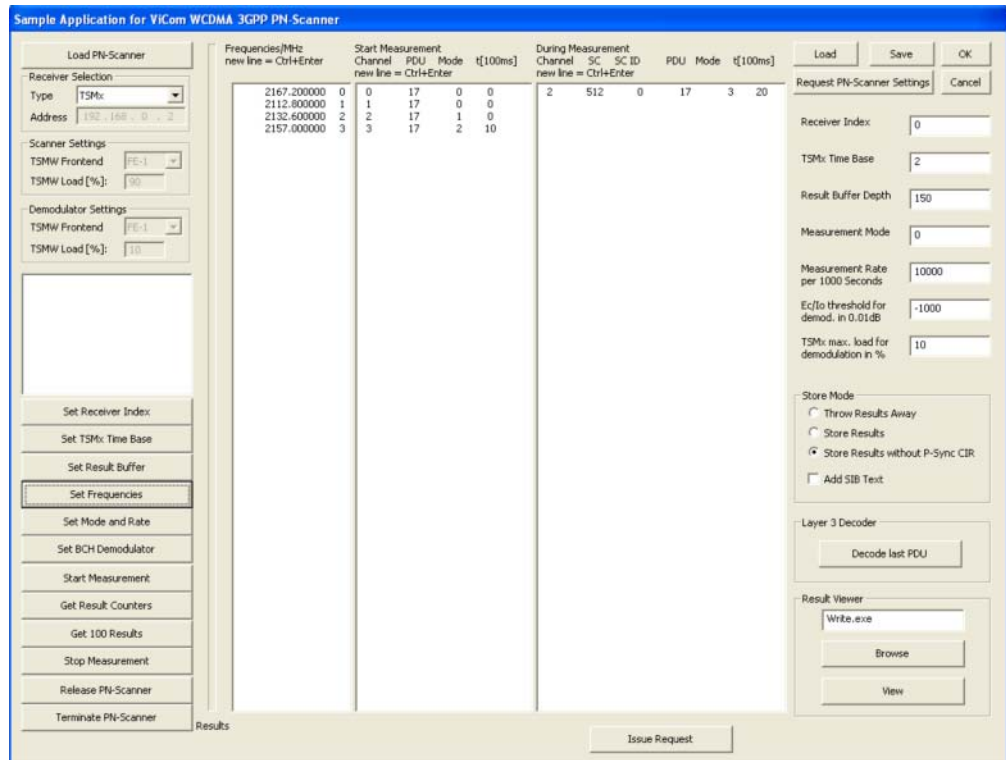
The sample application is supplied in both Release and Debug versions. In this document, it is assumed that the Release version is being used. The code for the sample application is found in the directory named "SampleForXXX" where XXX describes a particular ViCom interface.

The application and its user interface is rather designed to test the ViCom interface functions than to give an example for a measurement application. So it is possible to call interface functions at any time from the GUI, and it is possible to set values out of range to check the responses and the behaviour of the PN-Scanner in different situations.

The test application, SampleForViComWcdma3GppPnsRelease.exe can be found in the following location:

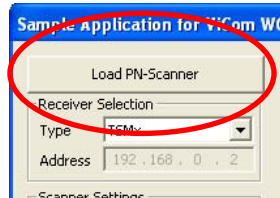


Ensure that ROMES demo is not currently running, and double click on the executable file SampleForViComWcdma3GppPnsRelease.exe. There is only one screen in the test application, shown below.



5.2.1 Setup connection

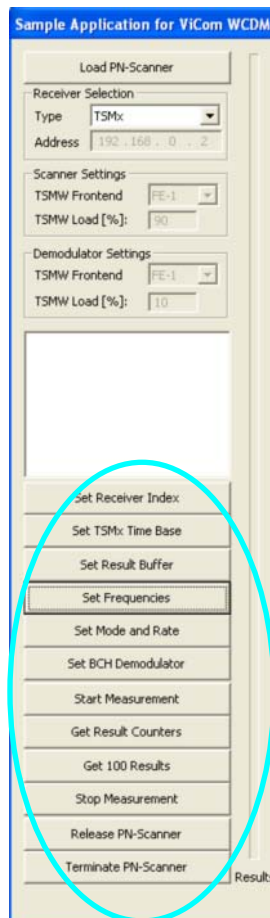
The first action after starting the sample application, should be to click the “Load PN Scanner” button, highlighted **below**. Before that, the device should be chosen that shall be used, i.e. if it is a TSMx or TSMW device.



If the PN Scanner is connected to a device and this is switched on as described above, then clicking “Load PN Scanner” will cause the sample application to load a ViComWCDMA3GppPns interface (See section 3 “Programming with the ViCom Interface” for more explanation about loading the interface). When the scanner is loaded successfully, information similar to that shown in the figure above, will appear in the text box below the “Load PN Scanner” button.

5.2.2 Setup a measurement command

The interface functions can now be accessed, by clicking on the buttons highlighted in the ring below.



The “Set..” buttons correspond to interface functions that set parameters in the scanner. An initial set of default values will appear in the settings fields (circled in ring 3, and the two long fields in the centre of the window). These can be edited in the fields, and sent to the scanner by clicking the corresponding button.

Note that the PN scanner has its own set of default values for all parameters except the frequencies. The PN scanner will use its own default values, unless they have been overridden by interface function calls using the dialogue buttons. Because there is no default value for the frequency in the PN scanner, at least one frequency must be set using the “Set Frequencies” dialogue button, before any measurements can be taken.

The PN-Scanner settings remain in the scanner until new interface functions are called, and the only exception to this is the settings for the BCH demodulator. These settings are channel specific and set to default values (PDU 17 (SIB 3) for each channel) when frequency settings are changed (see below).

Other buttons in the figure above correspond to interface functions that deal with measurements, and with unloading the scanner. “Start Measurement” starts the scanner measuring. “Get Result Counters” causes the current number of measurement results in the ViCom interface buffer to be displayed, as well as the number of measurements that have been deleted if the buffer has overflowed.

“Get next 100” causes the application to get the next 100 measurement results from the interface buffer. Results are either stored or thrown away, according to the radio buttons in the Store Mode box. If stored, the results will be written to the file ViComMeasurements, in the sub-directory \Application\LogFiles.

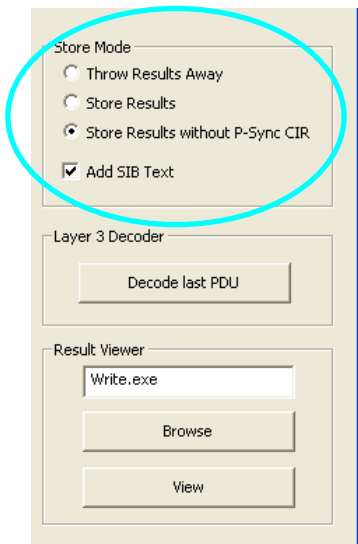
The figure below shows the two central text boxes in the GUI. In the left hand space, can be entered the frequency or frequencies at which the PN scanner should search, together with a number that is the local channel (frequency) identifier for the measurements. Please note, that although a default value appears in the GUI, this is not entered in the scanner until the “Set Frequencies” button has been clicked.

CDMA 3GPP PN-Scanner											
Frequencies/MHz		Start Measurement				During Measurement					
new line = Ctrl+Enter		Channel	PDU	Mode	t[100ms]	Channel	SC	SC ID	PDU	Mode	t[100ms]
new line = Ctrl+Enter		new line = Ctrl+Enter				new line = Ctrl+Enter					
2167.200000	0	0	17	0	0	0	429	0	15	3	0
2112.800000	1	0	15	1	0						
2132.600000	2	1	17	0	0						
2157.200000	3	2	17	1	0						
		3	17	2	10						
		3	15	2	0						

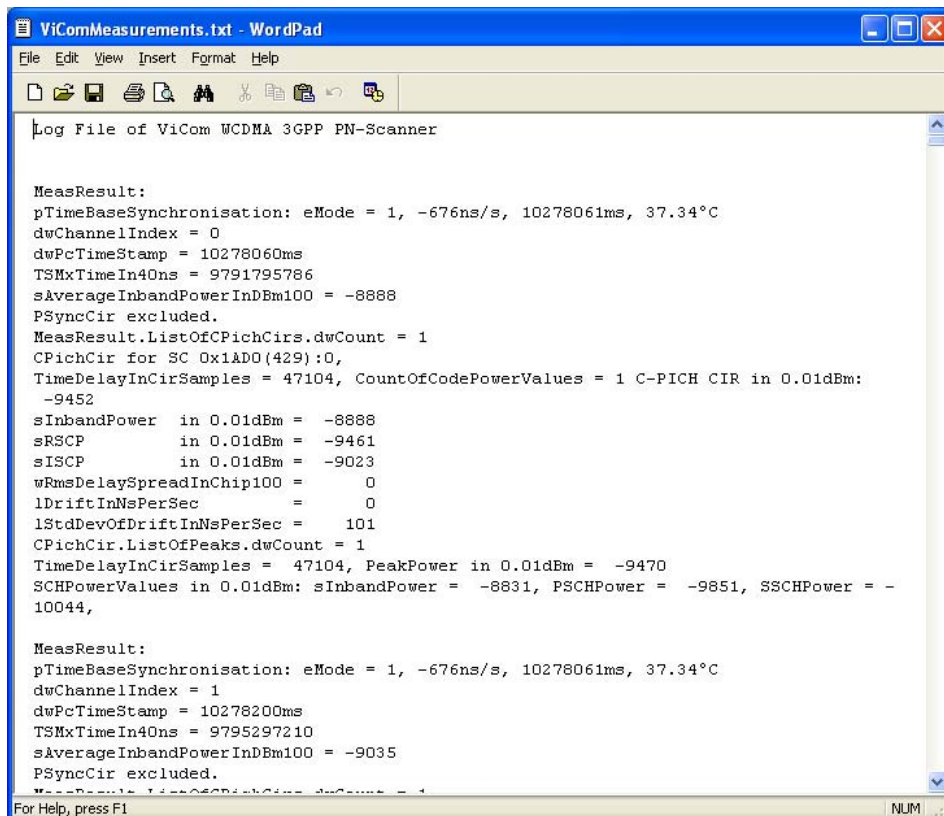
In the right hand space, can be entered the PDU values that correspond to the SIBs that the R&S TSMx should demodulate, if this option is present. Each PDU value must have a corresponding channel identifier, that must refer to one of the frequencies entered in the left hand box. Please see Appendix D “UMTS Technical Notes” for more information about UMTS SIBs, and a table of PDU values corresponding to each SIB. Please refer to the section BCH Demodulation on page 51.

5.2.3 Save and Load

Stored results can be browsed and viewed using the dialogue in the View Results box. In the text field, type the name of your preferred text editor, for example “notepad.exe” (the “Browse” button can be used to find the program if need be). Click the “View” button, and the sample application will open the result file using the preferred text editor.



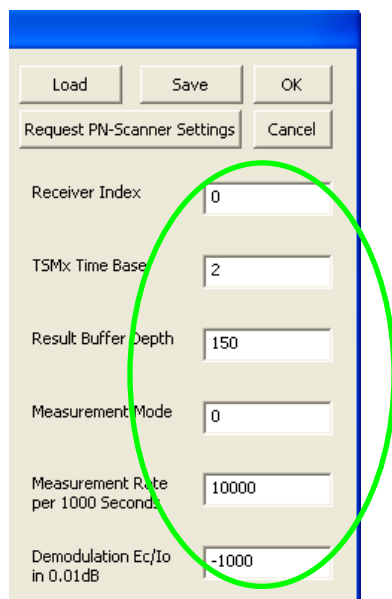
A sample of a R&S TSMx/TSMW measurement file is shown below. For an explanation of the fields listed, please see section 5, ViCom data fields, and the SMeasResult structure.



5.2.4 Update GUI with current scanner settings

The “Request PN-Scanner Settings” button highlighted below displays all PN-Scanner settings which apply at that moment to the scanner. This button can be clicked after starting a measurement, and will display the settings in the appropriate dialogue fields. The settings can also be stored in an R&S TSMx/TSMW settings file (*.tsm), and later reloaded, using the “Save” and “Load” buttons respectively.

There is also a function to force a reset of the PN-Scanner interface which has to be used if the R&S TSMx/TSMW is not responding due to longer disconnection or due to reset or power off during PN-Scanner operation.



5.2.5 Miscellaneous

The “Cancel” and “OK” buttons will terminate the application. “OK” will store the settings in the file “WCDMA3GPP_PnScannerSettings.bin” of the application directory. If available this file is used to initialise the application, (not the scanner!), after reloading.

If new R&S TSMx devices are connected to the Firewire or a R&S TSMx device is switched off then the interface must be reloaded. Please note, that although short interruptions of the IEEE1394 connection are tolerated, the R&S TSMx devices must not be switched off, disconnected or re-ordered while the interface is loading or loaded.

To see the code of the test application in MS Visual Studio 2003, open the project file, SampleForViComWcdma3GppPns.vcproj, in the folder ...\\SampleForViComWcdma3GppPns. To see the code of the test application using MS Visual Studio 2005, open the project file SampleForViComWcdma3GppPnsVS2005.vcproj.

5.3 BCH Demodulation

One central feature of the TSMx/TSMW series is the possibility to decode the messages sent on the BCCH-BCH for each cell that is encountered during the scan process. These messages contain information about the cell and various radio parameters available, besides information on specific capabilities a cell offers to the UE. How this demodulation process is controlled is subject of this chapter.

This section refers to the general description of the demodulation process in chapter “Using the Demodulators”. Please be sure to read that chapter first before starting with this section.

Below you can find a description of the way how the messages are obtained. It is a technically detailed description of the demodulation process. The second part shows how to use the ViCom API to get BCCH-BCH messages. Therefore, the sample application delivered with the API is shown and as the code of demo program is analyzed and explained. Common pitfalls are shown in the last section to help you to avoid those.

5.3.1 Measurement Details

The BCH demodulator is part of the WCDMA PN Scanner interface in ViCom. It has to be configured before a scan measurement is started. The configuration defines which information shall be demodulated and how the decoding process is controlled.

In the measurement configuration, for each channel a list of PDUs can be defined that shall be decoded. When the measurement is started, the scanner analyses the signals that are found and extracts information about the Node-Bs on air. The demodulator then first tries to synchronize on the master information block sent. After the MIB has been decoded, the remaining frames are analyzed.

The frames contain parts of or complete System Information Blocks. For each Node B that is found, the frame data is accumulated until all required frames have been found to start the demodulation of that SIB, if enabled.

To separate frames from different cells, the Scrambling Code is demodulated from the incoming signal and used as a key to store the PDUs in different frame containers. Since the SC is not a unique identifier, the system uses an additional indicator if the Scanner is not sure if a SC belongs to an already measured cell or not. This indicator increases each time such an uncertainty makes it necessary to split the data.



This uncertainty can be resolved by decoding the SIB 3 and investigating the Cell ID. Refer the demo code discussed below how to do this.

In the figure below, an example of that process is shown. Let's assume that it was requested to decode SIB 1, 3, 11 and 13. The upper stream is sent by a Node-B that has the SC 312, the lower one uses a SC of 13.

The BCH demodulator then extracts and saves the PDUs that were configured, each in the appropriate container of the Scrambling Code. Both scrambling codes 312 and 13 are found for the first time, so the aforementioned indicator is set to 0 for each (see the number after the scrambling code in the figure).

Once all required information is available, a decoding result is generated and made available to the caller. In the example, after the first 8 frames have been dispatched, SIB 13, 1 and 3 are ready to be demodulated for the Node-B with SC 312. To completely decode the SIB 11, some elements are still missing (these are marked as red element).



Note that the Master Information Block is always decoded, since it is necessary to get information from the MIB to decode other SIBs. In some networks, the Scheduling Information Blocks also contain data that is required to decode the SIBs correctly. They are implicitly decoded as well, if required.

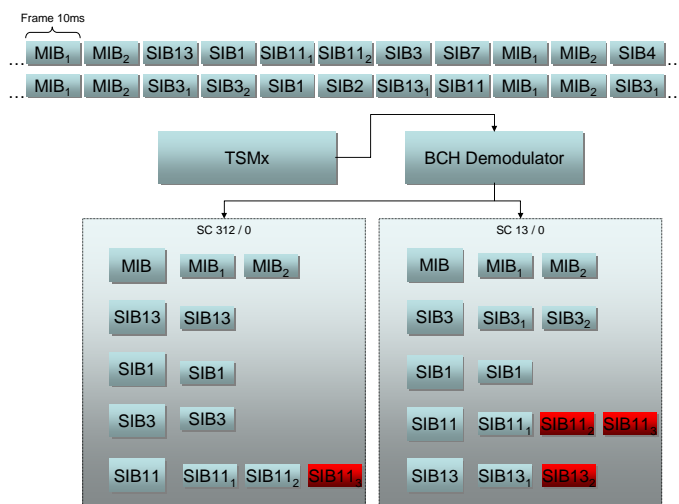


Figure 7 BCH Demodulator Measurements

5.3.1.1 Demodulation Modes

The demodulation settings described in the general chapter apply to the WCDMA demodulation as well, but there are some special constraints when it comes to demodulation of the WCDMA messages. These are described below.

The MIB handling differs from the default handling two ways:

- The MIB cannot be requested explicitly, even if it was defined with PDU_DEMOD_ON_CMD.
- If the MIB was configured to be reported on request only, it will be reported as soon as another SIB is reported or requested.

Since this handling is sometimes difficult to understand, we recommend not to change the default settings unless you definitely know what the modifications mean.



Due to internal architectural reasons, it is required to request the SIB 3 for a channel before other results might be returned. So make sure that for every configured channel, at least the SIB 3 is in mode PDU_DEMOD_ONLY_ONCE or PDU_DEMOD_REPETITION. If the mode is set to PDU_DEMOD_ON_CMD, other results will be returned after the first demodulation request for that SIB 3 has been performed successfully.

This also applies to the MIB.

5.3.1.2 Performance Measurements

The measurement rate at which decoding takes place is a crucial aspect in the BCH demodulation module. It was designed to be highly efficient and producing best results under various conditions. The actual demodulation probabilities are shown in this section.

Laboratory Measurements

One important aspect of the decoder quality is its block error rate. This block error rate tells one how many packets could not be decoded compared to how many packets were received in total.

To do a real-world performance measurement, the following test scenario was used: A TSMU received two signals, one from a Rohde & Schwarz CMU that acted as the data sender and a SMIQ device used to simulate some noise. Both were connected using a multiplexer to connect to the TSMU.

The E_c/I_0 threshold (P-CCPCH) was required to be at most -18 dB. With these constraints, 17915 PDUs were decoded correctly compared to 18045 received packets. That results in a block error rate of 0,72%, which is a reasonable value.

Additionally, measurements for lower E_c/I_0 were made. The CMU signal was decreased by 2 resp. 3 dBm to produce the results summarized in the table below:

Table 2 Laboratory BLER Results for CMU/SMIQ Test Environment

E_c/I_0 (P-CCPCH)	BLER
-18dB	0.72 %
-20 dB	40%
-21 dB	75%

Field Measurements

To get results under drift, additional measurements were done in a real network environment. The results are shown in Table 3 below.

The results were produced in an urban area with a maximum delay of 2 seconds between the E_c/I_0 measurement and the demodulation attempt. It can be observed that with E_c/I_0 falling below -11 dB, the demodulation probability decreases below 80%. With an E_c/I_0 value of -13 to -14 2/3 of the demodulation attempts still succeed, which should result in a fairly stable demodulation still.

Table 3 Field Measurement Results

E_c/I_0 [dB]	# Attempts	# Successful Attempts	Success Rate
-1..-2	4	4	100 %
-2..-3	66	66	100 %
-3..-4	338	336	99.4 %
-4..-5	766	758	99.0 %
-5..-6	832	812	97.6 %
-6..-7	815	779	95.6 %
-7..-8	780	727	93.2 %
-8..-9	727	652	89.7 %
-9..-10	750	654	87.2 %
-10..-11	704	582	82.7 %
-11..-12	834	658	78.9 %
-12..-13	748	545	72.9 %
-13..-14	907	607	66.9 %
-14..-15	1050	579	55.1 %
-15..-16	914	476	52.1 %
-16..-17	876	312	35.6 %
-17..-18	966	271	28.1 %
-18..-19	1029	220	21.4 %
-19..-20	1180	256	21.7 %
-20..-21	1070	96	9.0 %
-21..-22	958	70	7.3 %
-22..-23	767	61	8.0 %
-23..-24	617	25	4.1 %
-24..-25	496	27	5.4 %
-25..-26	305	15	4.9 %
-26..-27	150	7	4.7 %
-27..-28	76	6	7.9 %
-28..-29	44	1	2.3 %

5.3.2 Sample Application

In the SampleForViComWcdma3GppPns.exe application it is possible to experiment with the usage of the WCDMA ViCom API to demodulate the BCCH-BCH channel and decode the demodulated PDUs. A screenshot is shown below.

In this section, a complete walk-through is shown to help you getting a first demodulation result. So if you follow the steps shown below, you should get the BCH demodulation up and running very fast and also gain an impression of how to use the API.

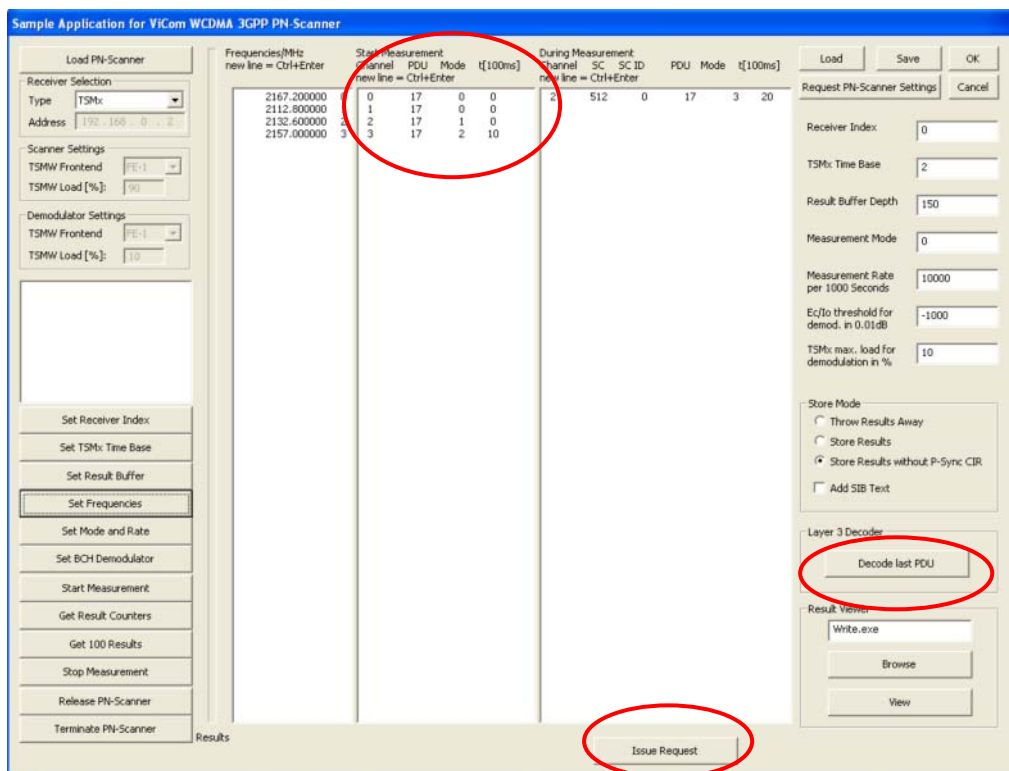


Figure 8 Screenshot of Demo application

The important parts of the demo application concerning the BCH demodulator are the center and right-handed columns and the “Add SIB Text” checkbox. For the case that dynamic requests shall be issued during the measurement, the Issue Request button can be used.

Once the application has been started, the following operation sequence should be performed to do a first sample measurement:

- Load a PN Scanner device (a TSMx/TSMW). This can be done by clicking on “Load PN Scanner” button. If this command was successful, the list box is filled with some details of the device.
- Click the buttons “Set Receiver Index”, “Set TSMx Time Base” and “Set Result Buffer” to do a proper setup of the standard settings in the TSMx. These set the values in the Text Fields next to “Receiver Index”, “TSMx Time Base” and “Result Buffer Depth” in the TSMx device.

- Define the frequency table and set the table using the “Set Frequencies” button. In the table, space-delimited pairs of a frequency and a channel index can be defined. For a simpler editing, it is also possible to add the frequencies only, one per line (a new line is started with Ctrl+Enter). The channel indices will be added automatically when “Set Frequencies” is pressed. For example, the mid-frequencies 2167.2, 2112.8, 2132.6 and 2157.2 MHz can be entered. These are the downlink frequencies of the UMTS channels in Germany, if you perform tests in another county, enter the according values.

new line = Ctrl+Enter	Chann	new lnr
2167.200000	0	0
2112.800000	1	1
2132.600000	2	2
2157.200000	3	3

Figure 9 Sample Frequency Settings for Germany

- As next step press “Set Mode and Rate” button. This applies the values stored in “Measurement Rate” and “Measurement Mode” Text Fields (shown on the right side) to the internal data model.
- Mark the checkbox “Add SIB Text” and make sure that the Store Mode is set to one of the “Store Results” radio buttons. Otherwise no demodulation result will be stored in the output file.

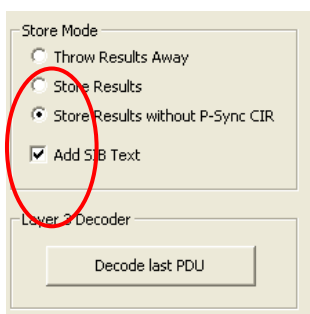


Figure 10 Layer 3 Decoder Options in Sample Application

- Specify the demodulation settings. Each line in the central list box contains demodulation settings for one channel/PDU pair. The numbers in the columns have the meanings defined below:
 - The first number is the channel number that can be seen in the frequency list box. It is used a short-cut to avoid handling with frequencies all the time, which is much more error-prone.
 - The second number is the internal PDU name. Refer to the CVicomWcdma3GppPnsInterfaceData::etPDU enumeration in the reference to find the mapping from SIB to PDU.
 - The third number is the demodulation mode. A description of the different modes can be found in the section Demodulation Mod on page 52.
 - The fourth and final column contains a timeout, if the demodulation mode is set to 2 (which means repetition mode). The value here is entered in 100 milliseconds. For example, a value of 10 defines 10 * 100 ms = 1 second as time-out.

To decode all SIB 3 for all four channels, enter the values shown in the screenshot below. Note that the SIB 1 of the third channel (channel index 2, PDU 15) will not be decoded from the start, but only after a request (see description below in the section “Issuing a Request during Measurement”).



Please note that all channels have the SIB 3 configured to be decoded at least one time. This is necessary to get any result.

Scanner			
Channel	PDU	Mode	t[100ms]
0	17	0	0
1	17	0	0
2	17	0	0
2	15	1	0
3	17	2	10

Figure 11 Channel / PDU demodulation settings

- Start the measurement with the equally named button. If you now click on “Get Result Counters” button regularly, you will find that new data is collected after the measurement has started.
- Fetch the results with “Get 100 Results”, so that if one of the store options was selected and the “Add SIB Text” checkbox was marked. Display the output file (using “View” button). The file is stored in the ViCom root dir in the file “Application\Logfiles\ViComMeasurements.txt”. In the opened text file, there should be some decoded SIB 3 message that was defined above, if the demodulation could be done within the 100 results.
You can search for L3 in the file to find demodulation results faster.

```

TimeDelayInCirSamples = 2980, PeakPower in 0.01dBm = -8429
SCHPowerValues in 0.01dBm: sInbandPower = -7622, PSCHPower = -8909, SSCHPower = -9043,
TimeDelayInCirSamples = 2984, PeakPower in 0.01dBm = -9495

MeasResult:
dwChannelIndex = 3
dwPcTimeStamp = 87955043ms
PduResult: SC 0x1030(259):0 PDU14 with 133 bits
0000100000011001000000011010011101000000000100000010001000011100000001000000100100010
011000000111101100101000001101011000010001100011
MeasResult.ListOfCPichCirs.dwCount = 0

L3DecoderResult: value MasterInformationBlock ::= { mib-ValueTag 2, plmn-Type
gsm-MAP : { plmn-Identity { mcc { 2, 6,
2 }, mnc { 0, 7 } } }, sibSb-
ReferenceList { sibSb-Type sysInfoType1 : 66, scheduling
{ scheduling { sib-Pos rep16 : 1 } } },
{ sibSb-Type sysInfoType2 : 4, scheduling { scheduling
{ sib-Pos rep8 : 0 } } }, { sibSb-Type
sysInfoType3 : 2, scheduling { scheduling
sib-Pos rep32 : 2 } } }, { sibSb-Type sysInfoType7 :
NULL, scheduling { scheduling { sib-Pos rep8
: 0 } } }, { sibSb-Type sysInfoType18 : 3,
scheduling { scheduling { sib-Pos rep16 : 1
} } }, { sibSb-Type sysInfoTypeSB1 : 4, scheduling
{ scheduling { sib-Pos rep64 : 31 } } }
} }

MeasResult:
dwChannelIndex = 3

```

Figure 12 Sample Result of BCH Demodulation

Issuing a Request during Measurement

In the example described above, the SIB 1 of the third channel is not included in the result file. To demodulate that PDU, it is now necessary to request that data explicitly. Follow those steps to accomplish this task.

It is assumed that the configuration is used for the example above. If you used different settings, please adapt these accordingly.

- Open the result viewer and search for the MIB of channel 2. If there is no MIB in the result file, try to get another 100 results into the file, until there is one in the file. Using the Scrambling code of that cell, the data for a specific demodulation request can be put together.
- Enter the request details in the right-most list box. Six columns have to be specified, which have the following meanings:
 - The first column defines the channel
 - The second and third columns define the scrambling code of the Node B that shall be demodulated. The first value is the scrambling code (0 to 511), the second the iteration counter.
 - The fourth column holds the PDU to decode, which must be the same as one of the PDUs defined in the centered list box with demodulation type 1 (DEMOD_ON_CMD)
 - The demodulation mode is entered in the fifth column. A good value is 3 for a Node-B specific demodulation, although any of the commands defined in the section During the Measurement can be put here.
 - If a time-out shall be specified, this can be done in the last column. In the example below, the time-out is 2 seconds (20 * 100 ms = 2000 ms = 2 s).

During Measurement					
Channel	SC	SC ID	PDU	Mode	t[100ms]
new line = Ctrl+Enter					
2	429	1	17	3	20

Figure 13 Specific Demodulation Request Configuration

- Click on the button “Issue Request” to send the request to the demodulator.
- Click on “Get 100 Results” and check the result file as described above for the demodulated string that has been requested. Be sure to close a previously started instance of your file viewer (default: write.exe) if the editor does not support automatic file updates (write.exe does NOT support that feature).

When you finished working with the demo application, don't forget to stop the measurement and release the resources of the TSMx/TSMW.



If you run another application that also accesses the same physical TSMx/TSMW device as PN Scanner, trying to initialize the ViCom in the second application will fail. So be sure not have other programs are open that use the WCDMA ViCom interface in parallel.

5.3.3 Programming Sample

To demonstrate how to use the API, a sample code snippet is shown below. In that sample, the Cell ID information is extracted from the SIB 3 message of the UMTS downlink channels in Germany, to show a simple case of what can be done with the BCH demodulator.

You can use this sample as base for you own programs. To make this sample run, it is necessary to have the ViCom Interfaces in the Include Path and to execute the application in the Application dir of the ViCom distribution to have all libraries available.

5.3.3.1 Code Listing

```

#include "stdafx.h"

#include <windows.h>
#include <iostream>
5  #include <conio.h>

#include "ViComWcdma3GppPnsInterface.h"
#include "ViComLoader.h"

10  using namespace std;

/*****

#define EXEC_VICOM( msg, call ) \
15  cout << msg << "..."; \
    call; \
    if ( cError.GetErrorCode() > 0 ) { \
        char* szErr = new char[strlen(cError.GetErrorString()) + 1]; \
        strcpy(szErr,cError.GetErrorString()); \
20  throw CViComError(cError.GetErrorCode(), szErr); } \
    cout << "ok" << endl;

/*****

25  void RunBchDemodulationSample()
    {
        CViComError cError;
        CViComLoader< CViComWcdma3GppPnsInterface > cBchDemodulator;

30  //
        // Below is the standard initialization sequence of a ViCom module.
        //
        EXEC_VICOM( "Loading PN Scanner",
35  cBchDemodulator.LoadTsmx( cError ) );

        EXEC_VICOM( "Retrieving special interface",
            CViComWcdma3GppPnsInterface* pcPnScanner = cBchDemodulator.GetpInterface(
                cError ) );

40  //
        // Common initialization sequence
        //
        EXEC_VICOM( "Setting receiver index",
45  pcPnScanner->GetBasicInterface().SelectReceiver( cError, 0 ) );

        EXEC_VICOM( "Setting time base",
            pcPnScanner->GetBasicInterface().SetTimebaseSynchronisationMode( cError,
                CViComBasicInterfaceData::TIME_BASE_SYNC_MODE_PPS_WATCHED_BY_GSM ) );

50  CViComBasicInterfaceData::SResultBufferDepth cResultBufferDepth;
        cResultBufferDepth.dwValue =
        CViComBasicInterfaceData::SResultBufferDepth::dwMax;

```

```

EXEC_VICOM( "Setting result buffer",
55     pcPnScanner->GetBasicInterface().SetResultBufferDepth( cError,
        cResultBufferDepth ) );

    //
    // The measurement rate and mode must be specified to produce a
    // meaningful measurement result.
60     // Set measurement mode to the smallest common divisor, which is HIGH SPEED
    //
    CViComWcdma3GppPnsInterfaceData::SMeasurementRate cMeasRate;
    cMeasRate.dwValuePer1000Sec =
65     CViComWcdma3GppPnsInterfaceData::SMeasurementRate::dwMaxPer1000SecForTSML_HIGH_S
PEED;
    EXEC_VICOM( "Setting Measurement Details",
        pcPnScanner->SetMeasurementMode( cError,
            CViComWcdma3GppPnsInterfaceData::MEAS_MODE_HIGH_SPEED, cMeasRate ) );

70     //
    // Now a set of frequencies is defined on which a BCH may be found. This
    // might
    // vary in different countries and for different network providers, so the
    // value can be changed easily.
75     //
    // Note that depending on the device attached to the system, the maximum
    // number
    // of frequencies differs. For the TSML-W, it is 6, for example.
    //
80     // Frequencies shown here are valid for Germany.
    //
    static double FREQUENCIES_IN_MHZ[] = {
        2167.200000, // T-Mobile D
        2112.800000, // Vodafone
85         2132.600000, // E+
        2157.200000 }; // O2

    CViComWcdma3GppPnsInterfaceData::SChannelSettings cFreqSettings;
    cFreqSettings.dwCount = sizeof FREQUENCIES_IN_MHZ / sizeof( double );
90     cFreqSettings.pdTableOfFrequencyInMHz = FREQUENCIES_IN_MHZ;

    EXEC_VICOM( "Setting Frequency Table",
        pcPnScanner->SetFrequencyTable( cError, cFreqSettings ) );

95     //
    // Now the demodulation details can be defined. In this sample, different
    // decoding strategies are shown that are supported by the ViCom demodulator.
    //
    CViComWcdma3GppPnsInterfaceData::S_PDU_Requests::S_PDU_Request PDU_REQUESTS[]
100 =
    {
        // Decode SIB 3 only once for channel 0
        { 0, CViComWcdma3GppPnsInterfaceData::PDU_FOR_SIB3,
            CViComWcdma3GppPnsInterfaceData::PDU_DEMOD_ONCE, 0, 0, 0 },
105
        // Decode SIB 1 of channel 0 only on command (see below for the command
        // that issues the request)
        { 0, CViComWcdma3GppPnsInterfaceData::PDU_FOR_SIB1,
            CViComWcdma3GppPnsInterfaceData::PDU_DEMOD_ON_CMD, 0, 0, 0 },
110
        // Decode SIB 3 of channel 1 only once
        { 1, CViComWcdma3GppPnsInterfaceData::PDU_FOR_SIB3,
            CViComWcdma3GppPnsInterfaceData::PDU_DEMOD_ONCE, 0, 0, 0 },
115
        // Decode SIB 3 of channel 2 endlessly without pause, i.e. if a SIB 1 is
        // detected, it is demodulated
        { 2, CViComWcdma3GppPnsInterfaceData::PDU_FOR_SIB3,
            CViComWcdma3GppPnsInterfaceData::PDU_DEMOD_REPETITION, 0, 0, 0 },
120
        // Decode SIB 3 of channel 3 every second
        { 3, CViComWcdma3GppPnsInterfaceData::PDU_FOR_SIB3,
            CViComWcdma3GppPnsInterfaceData::PDU_DEMOD_REPETITION, 10, 0, 0 },
    };

```

```

125     CViComWcdma3GppPnsInterfaceData::SBchDemodulationSettings
        cBchDemodulationSettings;
        cBchDemodulationSettings.lEcToIoThresholdInDB100 =
CViComWcdma3GppPnsInterfaceData::SBchDemodulationSettings::lMinEcToIoThresholdIn
DB100ForTSM; // == -10 dB
130
        cBchDemodulationSettings.sStartMeasurementRequests.dwCountOfPDURequests =
        sizeof PDU_REQUESTS / sizeof
CViComWcdma3GppPnsInterfaceData::S_PDU_Requests::S_PDU_Request;
        cBchDemodulationSettings.sStartMeasurementRequests.pPDURequest =
135     PDU_REQUESTS;

        EXEC_VICOM( "Configuring BCH demodulator",
                    pcPnScanner->SetBchDemodulationSettings( cError, cBchDemodulationSettings
) );
140
        EXEC_VICOM( "Start measurement",
                    pcPnScanner->GetBasicInterface().StartMeasurement( cError ) );

        //
145     // The decoding is done until the user manually quits the decoding
        // process by pressing some key.
        //
        cout << endl << "Measurement is active, press any key to stop it";

150     CViComWcdma3GppPnsInterfaceData::SMeasResult* pcResult = NULL;
        while ( ( pcResult = pcPnScanner->GetResult( cError, 1000 ) ) )
        {
            // Check for keyboard stroke to exit demo
155         if ( _kbhit() )
            {
                break;
            }

            if ( ! pcResult->pPduResult )
160         {
                cout << ".";
                continue;
            }

165         cout << endl << "Channel " << pcResult->dwChannelIndex;
            cout << " SC " << pcResult->pPduResult->ExtendedSC.wSC;
            cout << "/" << pcResult->pPduResult->ExtendedSC.wIndicator ;
            cout << " PDU " << pcResult->pPduResult->ePDU << endl;

170         //
            // If MIB of channel 2 has been decoded, we issue a separate request to
            // decode the SIB 3 of that channel. This is possible because the SIB 3
            // of channel 2 has been configured to be decode-able on command
            //
175         if ( pcResult->dwChannelIndex == 0
                && pcResult->pPduResult->ePDU ==
                    CViComWcdma3GppPnsInterfaceData::PDU_FOR_MIB )
            {
                //
180                // Decode SIB 1 of channel 0 now. This has to be done per Node-B!
                //
                CViComWcdma3GppPnsInterfaceData::S_PDU_Requests::S_PDU_Request
                    SINGLE_PDU_REQUEST[] =
185                {
                    { 0,
                      CViComWcdma3GppPnsInterfaceData::PDU_FOR_SIB1,
                      CViComWcdma3GppPnsInterfaceData::PDU_DEMOD_NODE_B, 0,
                      pcResult->pPduResult->ExtendedSC.wSC / 16 ,
                      pcResult->pPduResult->ExtendedSC.wIndicator }
190                };

                CViComWcdma3GppPnsInterfaceData::S_PDU_Requests cRequests;
                cRequests.dwCountOfPDURequests =
                    sizeof SINGLE_PDU_REQUEST /

```



```

195         sizeof
CViComWcdma3GppPnsInterfaceData::S_PDU_Requests::S_PDU_Request;
        cRequests.pPDURequest = SINGLE_PDU_REQUEST;

        EXEC_VICOM( "Issuing a special decoding command",
200         pcPnScanner->IssuePduRequests( cError, cRequests ) );
    }

    //
    // If we detect a SIB 3, we try to retrieve the CellID in the message.
205    //
    if ( pcResult->pPduResult->ePDU ==
CViComWcdma3GppPnsInterfaceData::PDU_FOR_SIB3 )
    {
        //
210        // Now that a PDU has been decoded, it is converted to an ASN.1
        // syntax compliant string representation. The input structure is
        // filled with a copy of the received PDU result bitstream, according
        // to the ViCom rule that returned data is only valid until the next
215        // interface method is called.
        //
        CViComWcdma3GppPnsInterface::SL3DecoderRequestData cL3PDU;
        cL3PDU.m_dwPDU          = (DWORD) pcResult->pPduResult->ePDU;
        cL3PDU.m_dwBitCount    = pcResult->pPduResult->dwBitCount;

220        DWORD nByteLen      = (pcResult->pPduResult->dwBitCount + 7) / 8;
        cL3PDU.m_pbBitStream = new byte[ nByteLen ];
        memcpy( cL3PDU.m_pbBitStream, pcResult->pPduResult->pbBitStream,
nByteLen );

225        EXEC_VICOM( "Decoding SIB 3",
                    CViComWcdma3GppPnsInterface::SL3DecoderResultData* pSL3_Data =
                    pcPnScanner->RetrieveTextForPDU( cError, cL3PDU, 30000 ) );

        delete[] cL3PDU.m_pbBitStream;
230        if ( ! pSL3_Data )
        {
            cout << "Could not decode SIB 3! Error " << cError.GetErrorCode() <<
                endl;
235            continue;
        }

        char* pszCellId = NULL;
        const char* pszSearchString = "cellIdentity[16 Bit] ";
240        if ( pszCellId = strstr( pSL3_Data->m_pcStringPDU, pszSearchString ) )
        {
            int nCellId = atoi( pszCellId + strlen( pszSearchString ) );
            if ( nCellId )
245                cout << "CellID: " << nCellId << endl;
        }
    }
}

250 EXEC_VICOM( "Stop measurement",
            pcPnScanner->GetBasicInterface().StopMeasurement( cError ) );

EXEC_VICOM( "Waiting for stop measurement signal",
255 pcPnScanner->GetBasicInterface().HasMeasurementStopped( cError, 1000 ) );

EXEC_VICOM( "Unloading PN Scanner...",
260 cBchDemodulator.ReleaseTsmx( cError, false ) );

/*****/

265 int _tmain(int argc, _TCHAR* argv[])
    {

```



```

    try
    {
        cout << "ViCom BCH Demodulation Demo" << endl << endl << endl;
270     RunBehDemodulationSample();
    }
    catch ( CViComError cError )
    {
        cout << cError.GetErrorString() << endl;
275     delete [] cError.GetErrorString();
        return cError.GetErrorCode();
    }
    return 0;
280 }

```

5.3.3.2 Code Structure

The sample code defined above works similar to the demo application and the steps described before. To improve readability, a macro is defined to execute a ViCom action and to convert an error into an exception. Additionally, the current action is logged to the standard console.

The purpose of the demo code is to find the SIB 3 of a set of frequencies and to decode the Cell ID information included in the message.

The main method is simply an exception-catch wrapper for the central function. The central function is not divided into smaller methods to show the required procedure in one row without referring from one method to the other.

The first steps are to load a WCDMA PN Scanner and to initialize it. In line 34 the PN Scanner module is loaded. The initialization process that configures the basic properties of the TSMx/TSMW device follows in lines 43 - 89. This also includes the definition of the frequency to channel mapping. The channel is implicitly defined by the position of the frequency in the given array (line 78).

The BCH demodulator is configured in lines 95-118. In this code piece, the PDUs demodulation settings are defined per channel. The settings are explained in the code and similar to the example of the demo application.

After the measurement is started (line 135), a loop is started that tries to receive a measurement result from the PN Scanner. If a result is available, it is checked for a PDU result (line 152). Before a check is done if the user has requested to quit the application (by pressing any key).

The result is queried as follows:

```
while ( ( pcResult = pcPnScanner->GetResult( cError, 1000 ) ) )
```

The returned value is NULL if no result is available within the defined timeout of 1000 milliseconds. The returned structure is valid until the next interface method is called.



Due to internal reasons, you should not quit the application using Ctrl+X. This does not allow the program to properly clean up the TSMx, which might lead to problems when trying to use the device again in the same windows session. If there are problems, a reboot should solve them.

When some PDU was found, the code checks if it is the indicator for a special request (since the SIB1 of channel 0 will requested on command, as defined in the configuration before). For the special request, a similar workflow is used as during the BCH configuration (lines 175-192).

One important thing to notice when a specific PDU for a Node-B shall be decoded is shown below. The Scrambling Code in the measurement result is the primary scrambling code of the cell. As input for the new request a short SC ranging from 0 to 511 is expected, so the primary scrambling code of the response has to be divided by 16. The indicator must be used as received.

```
CViComWcdma3GppPnsInterfaceData::S_PDU_Requests::S_PDU_Request
    SINGLE_PDU_REQUEST[ ] =
    {
        { 0,
            CViComWcdma3GppPnsInterfaceData::PDU_FOR_SIB1,
            (byte)CViComWcdma3GppPnsInterfaceData::PDU_DEMOD_N
            ODE_B, 0,
            pcResult->pPduResult->ExtendedSC.wSC / 16 ,
            pcResult->pPduResult->ExtendedSC.wIndicator }
    };
```

To fulfill the purpose of the program, a SIB 3 decoding result is parsed in lines 208-239. Therefore the received data is copied into a special layer 3 decoder result structure. This is necessary since one of the main principles in ViCom is that the return value of a function is valid only until the next ViCom interface method is called, which is true in this case.

As shown in the excerpt from the code below, the byte length of the answer has to be calculated before from the number of bits available in the stream. The value must be aligned to a number dividable by 8 to get the number of bytes required.

```
DWORD nByteLen          = (pcResult->pPduResult->dwBitCount + 7) / 8;
cL3PDU.m_pbBitStream    = new byte[ nByteLen ];
memcpy( cL3PDU.m_pbBitStream, nByteLen,
        pcResult->pPduResult->pbBitStream, nByteLen );
```

In the resulting string, a text with the prefix "cellIdentity[16 bit]" is searched, since according the ASN.1 this is the official name of that information element.

Finally, some resource clean-up is done.

5.3.4 Frequently Asked Question

This section explains some of the most common questions that arise when using the ViCom BCH demodulator. They don't have any particular order, so skip through them when you encounter a problem – maybe it's one of the problems other people already had.

- ? *Are there any other representation layouts other than the string format? Which ones are ASN.1 compliant?*
- ! The ViCom API returns a string formatted version of the data according to the formatting rules defined in ASN.1 specification. Additionally, some prefixes are used to distinguish different versions of the UMTS RRC message standard (r3 for Release 99, r5 or r6), when necessary.
There is no other meaningful form of the data returned from the ViCom library, except the raw byte-stream.
- ? *Is the LAC parameter in the SBchCellIdentification structure filled at any time?*
- ! It is only provided, if SIB 1 has been demodulated, which contains the LAC.
- ? *I don't get any results although I configured some PDUs to be demodulated. Even the MIB is not reported. What's wrong?*
- ! Please check that your configuration allows the reporting of SIB 3 for that channel. You can verify this by checking if a PDU 17 is set either to mode 0 or 2 in the demo application. If not, configure the SIB 3 to be decoded in your start measurement setting.
- ? *SIB10 and 16 are not enumerated in the ViCom etPDU. How can we select them (their corresponding PDUs) for decoding in ViCom, particularly SIB16?*
- ! SIB 10 is not demodulated because it is only used in FDD and has been omitted completely as of UMTS TS25.331 Rel. 5.
Currently SIBs which carry information in multiple occurrences are not demodulated in general. This is the reason, why SIBs 15.2, 15.3 and 16 have no related PDUs in the interface..
- ? *BCH E_c/I_0 threshold ($IEcToloThresholdInDB100$) is the E_c/I_0 threshold for which physical channel (P-CCPCH or CPICH)?*
- ! It is the E_c/I_0 of the P-CPICH. The data needed for demodulation are requested when the E_c/I_0 of the P-CPICH is exceeding the given threshold. The CPICH is used as the scanner monitors all Node Bs by means of C-PICH E_c/I_0 measurements and there is no extra measurement of the P-CCPCH E_c/I_0 during the monitoring phase. But both values should be in a fix ratio per node B anyway. And actually both the P-CPICH and the P-CCPCH signals will be used for demodulation, as the channel estimation as a part of demodulation is done by the P-CPICH.

- ? *What is the minimum configuration of decoded PDUs that have to be set to make the decoder work?*
- ! You must request at least SIB3. No SIB PDU result will be provided until SIB3 has been decoded, because the BCH Demodulator provides each SIB PDU together with CI (Cell Identity from SIB 3). E.g. if you request SIB 3 (= PDU 17) you will receive MIB, SIB3 and, if applicable, SB1 and SB2, after SIB 3 has been decoded by the BCH Demodulator.
- ? *Once a SIB has been decoded for a Node B, is it ever decoded again at a later point in time?*
- ! As long as the system is sure that the same signal is received, the first decoding result is considered valid and returned. If there is an uncertainty about the signal source (for the same SC on the same channel, different frame timing or a longer pause between the measurement points occurred), the decoding process is repeated. To notify about this event, the scrambling code indicator is incremented. To be sure to have the same Node B in the case that two results have been found with two different indicator values, the CellID must be decoded. This behaviour can be changed when configuring the measurement. For each Node B, it is possible to define another strategy how often the PDUs shall be decoded. It is also possible to reset the stored results cache.
- ? *Is there any effect in changing load for BCH demodulation in ROMES with TSMx in HiSpeed or HiDynamic mode?*
- ! No, the load is fixed to 10%. This was a quite reasonable value that was found out with the ROMES. With the NRE for the BCH demodulator demodulations with much higher priority may be requested. The scheduling is influenced in a way that the TSMx gets a maximum penalty in measurement rates on 10% between SIB decoding and not decoding.
- ? *Is there a way to adjust TSMx Sync. Rate in ViCom as it is possible in ROMES? If yes, how does this affect scheduling?*
- ! If the TSMx is used with ViCom new node Bs are searched on each measurement. As this procedure (WCDMA synchronisation) is done with the same RF signal section as all the other measurements (RSCP, CIR etc.) the synchronisation does not affect the load TSMx or cause additional the receiver usage. It affects only the processor load at the connected PC. The difference in processing one RF section with and without synchronisation was up to 30% in the phase. At present, with more CPU cache, this difference is less than 10%. So we decided to synchronise on each RF sample with ViCom and simplify ViCom usage by elimination of another parameter. By the way, we are able to synchronise to a Node-B in the high speed mode when it is visible for at least 1 ms.

6 ViCom GSM Network Scanner

The GSM network scanner functionality offers a whole bunch of measurement methods to monitor and analyze GSM networks. Therefore, high sophisticated algorithms have been integrated in the TSMx software. Nevertheless, the API was designed to be as simple as possible. A basic understanding of the things happening behind the scene is still necessary to make full use of the power the device offers to you.

This chapter explains that functionality. In order to make you familiar with the different measurement modes the network scanner provides, the basic principles of how the network scanner actually finds the data are explained in the first section. The next section explains the usage of the sample application provided with the API. A step-by-step example helps you to do the first measurements and interpret the result. The chapter is terminated by a small demo program that is explained to show how to use the network scanner API in the code.

6.1 Measuring GSM Signals

The API of the ViCom GSM Network Scanner (called GSM NWS in the rest of the chapter) provides a very flexible way to specify what kind of measurements and shall be done in what way. To use this flexibility efficiently and to understand what the different tasks are really doing, this section gives some detailed insights into the algorithms of the network scanner.



It is important to know that the GSM Network Scanner is capable of modifying the measurement tasks while the measurement is active. This allows to adapt the measurement specification to new environmental settings etc. For most of the other ViCom APIs this is not the case.

The single exception for this capability is that the list of channels that shall be scanned must be known upfront and cannot be changed during the measurement.

6.1.1 Network Scanner Specials

Before the measurement tasks are described, the following sections set the scene for the usage and explain some common elements of the ViCom GSM NWS API. Most of them deal with configuration and result interpretation.

6.1.1.1 Frequency Configuration

Before any measurement can be started, some basic configurations have to be made. The most important and general aspect of this configuration is the list of channels that shall be scanned. It is crucial to know that the list of these frequencies has to be sorted in ascending order. Below is a short sample for the configuration.

```

DWORD dwFrequencyCount = 5;
CViComGSMNWSInterfaceData::SFrequencySetting* ptFrequencySettings =
    new CViComGSMNWSInterfaceData::SFrequencySetting[dwFrequencyCount ];
memset( ptFrequencySettings, dwFrequencyCount *
    sizeof CViComGSMNWSInterfaceData::SFrequencySetting, 0 );

ptFrequencySettings[ 0 ].dCenterFrequencyInHz = 935200000;
ptFrequencySettings[ 1 ].dCenterFrequencyInHz = 935400000;
ptFrequencySettings[ 2 ].dCenterFrequencyInHz = 936600000;
ptFrequencySettings[ 3 ].dCenterFrequencyInHz = 940200000;
ptFrequencySettings[ 4 ].dCenterFrequencyInHz = 950000000;

ptFrequencySettings[ 0 ].bRequestAutoDemodulationOfST3 = true;
ptFrequencySettings[ 1 ].bRequestAutoDemodulationOfST3 = true;
ptFrequencySettings[ 2 ].bRequestAutoDemodulationOfST3 = true;
ptFrequencySettings[ 3 ].bRequestAutoDemodulationOfST3 = true;
ptFrequencySettings[ 4 ].bRequestAutoDemodulationOfST3 = true;

CViComGSMNWSInterfaceData::SChannelSettings cChannelSettings;
cChannelSettings.dwCount = dwFrequencyCount;
cChannelSettings.pTableOfFrequencySetting = ptFrequencySettings;

pcGsmScanner->SetFrequencyTable( cError, cChannelSettings );

```

In this configuration, five channels with the following center frequencies shall be scanned: 935.2, 935.4, 936.6, 940.2 and 950 MHz. The System Information Type 3 message shall be decoded for every single channel. For more details on this setting, refer to section “Demodulation of BCCH and System Type Information”.

6.1.1.2 Time Synchronisation



As time synchronisation mode, only the following subset of potential values is permitted when used in the GSM network scanner: Internal, GSM and PPS.

6.1.1.3 Measurement Scans

When the TSMx has measured all channels and gathered the results this is called a scan. One scan always covers the complete band specified by the frequencies configured before. These channels are measured and processed in groups of at most ten by the TSMx device due to internal reasons.

Since most of the TSMx devices can perform multiple different measurements in parallel, it might occur that one scan cannot be performed completely in one row. In such a case, another measurement task is processed before the scan is continued. A scan interrupted in that way is finished as fast as possible, but it might be the case that the API reports some results before the complete task is finished. It is even possible that some results of a not finished scan have to be thrown away and another scan is started. The missing parts are then measured as soon as there is time to do so.

The channel groups that are measured can contain up to 10 channels, where all channels must be within 2 MHz. This is done to use the capabilities of the TSMx hardware as efficiently as possible. If there is only a subset of channels to be measured, the group can contain less than 10 channels.

The better the groups are filled, the higher measurement rates are possible. The TSMQ can measure up to 100 channels per second when most of the groups are filled. For the TSML, the maximum reduces to about 20 channels per second.

6.1.1.4 Result Details

Some data in the result structure is not directly related to one single measurement task, but common to more than one or all. These are explained in the subsequent paragraphs.

Split Scans

One measurement result that is requested can contain many different types of results, depending on which measurement tasks are currently activated. Besides this, one measurement result must not necessarily contain all the data made in one scan (as explained above). It is possible that the data of one scan is not completely put into one result structure. Therefore, the result contains a scan index and a flag whether it is the last structure holding data for that scan index.

In the result structure, the `pdwScanCount` member is a pointer where number of the current scan is found. This is an increasing value, for each scan this will be incremented by one. With this attribute, the split results can be grouped again to form a complete scan result.

Frequency Index

The channels are referenced in the result structures as index that can be used to refer to the frequency settings table configured before the measurement is started. The index is called `wFrequencyIndex` inside the structures.

List of Executed Measurements

In some cases, there is no frequency index given directly. For these results, there is a way to match the result data onto the frequencies specified in the `ListExecutedMeasSpec` structure of the result. This list provides the frequency indices of all measured data in the result. Note that the order is important here when it comes to match the indices to the result data. Details are explained in the related sections.

6.1.2 Measurement Tasks

In this section, the different types of measurement tasks that the GSM network scanner can handle. While each task is described, short code fragments are shown that demonstrate the usage of the API in more detail.

6.1.2.1 Power Measurement

This measurement is the one which is enabled per default for every frequency that is being measured. It cannot be disabled and will deliver results in every scan. One scan contains a complete set of power values for all configured channels.

The GSM network scanner is capable of measuring different types of power. Which type is actually used for a power value is specified as part of the result and explained below.

Total Channel Inband Power

This is the default way of measuring power of a channel. After a channel has been measured for about 50 ms, the TSMx searches for the part of the signal that covers highest power and lasts as long as a timeslot.

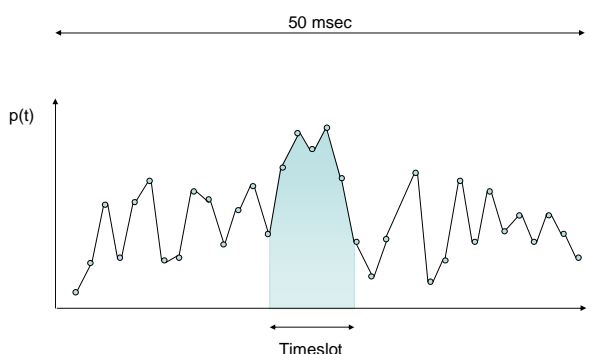


Figure 14 Total Channel Inband Power Calculation

Power of the SCH

When the scanner is synchronized on a channel, it can measure the SCH power (time-slot 0). There are several modes how the SCH power can be measured, which are also returned in the result and described below:

- ETS only: The extended training sequence burst has been found and the power of burst is returned.
- SCH Demodulation: An SCH demodulation attempt was successful and the power value is derived from that burst. Once such a demodulation was successful, the BTS found is managed in so-called Active Set containers. When a later SCH demodulation fails, it is then possible to relate the signal to a formerly demodulated BTS using cross correlation algorithm.

The SCH of a BTS is demodulated successfully if the ratio of the SCH power to the total inband power is greater than -6 dBm. There are two Active Set containers that are used in different ways.

- Active Set 1: The SCH signal has been demodulated in the last 5 minutes. The drift of the incoming signal is within $-16\mu\text{s}$ and $+35\mu\text{s}$ to be able to process reflected and faded signals as well. The incoming signal must then have $P_{\text{SCH}}/P_{\text{tot}}$ of at least -11 dBm.
- Active Set 2: The SCH signal has been demodulated in the last 30 seconds. The drift of the incoming signal is within a range of $\pm 6\mu\text{s}$. The power ratio must not be less than -9 dBm in that case to perform a correlation to the original BTS.

The code below is an example how the maximum power value in a scan result can be found. The result normally contains a whole bunch of that power values.

```
double dMaxPower = -9999.0;
long nFrequency = 0;
SViComList<CViComGSMNWSInterfaceData::SMeasResult::SPowerResult>::SLinkedObject*
pcPowerResult = pcMeasResult->ListPowerResults.pFirst;

while ( pcPowerResult )
{
    if ( dMaxPower < 0.01 * pcPowerResult->sPowerInDBm100 )
    {
        dMaxPower = 0.01 * pcPowerResult->sPowerInDBm100;
        nFrequencyIndex = pcPowerResult->wFrequencyIndex;
    }
    pcPowerResult = pcPowerResult->pNext;
}
cout << "Max pow: " << dMaxPower;
cout << " Frequency: " <<
ptFrequencySettings[nFrequencyIndex].dCenterFrequencyInHz
cout << endl;
```

6.1.2.2 SCH Measurement

Once a SCH has been demodulated, the GSM network scanner can report the content of the data send in the SCH burst. This data consists of the BSIC of the BTS and the frame number, in which that specific SCH was sent.

The information is especially important to start a specific demodulation request of System Type Information messages. Only after the scanner could successfully synchronize to as BTS, such a command can be answered. The data can be found in the `CViComGSMNWSInterfaceData::SMeasResult::SSCHInfoResult` structure and can directly be used to fill the request structure for a decoding request (see below).

6.1.2.3 Counting SCH Measurement Failures

Not every SCH demodulation attempt will be a successful one. For some applications, it might be useful to know how many failures occurred when SCH demodulation was requested. Therefore, the GSM network scanner reports the number of failures when the SCH measurement mode is active.

The power values derived from those attempts can be used as an estimation of the upper limit of the measured signal. Since the scanner already extracted all channel information it could from the incoming signal, the remaining signal power is at most the one from one or more remaining cells (+/- some noise).

The result values are returned in the same way as the default power measurements described above.

6.1.2.4 Demodulation of BCCH and System Type Information

Another measurement task allows the demodulation of the System Information Type messages sent on the BCCH. Before that can be done, the scanner must be synchronized on a BTS signal, i.e. there must be at least one successful attempt to demodulate the SCH burst.

There are two different ways of requesting the decoding from the scanner:

- Either during the initial configuration procedure. In the frequency table, a flag can be set that the decoding process shall be done automatically until the attempt was successful.
- Or during the measurement. Some more setup details have to be fixed in that case, time-out values for example.

Since the first option is relatively self-explanatory to use, we concentrate on the second option here. The configuration for a request is done using the structure

`SDemodulationSettings`.

Besides the information retrieved from the `SSCHInfoResult`, these settings must be defined for each message type (1 to 4):

- Shall a new request be started, an existing request be cancelled or shall the current settings be kept as it is.
- How long shall the network scanner try to satisfy the demodulation (if mode is set to `DEMODREQUEST_START_REQUEST`).
- An optional user-defined request id can be given to identify the request later in the result. This value is not used by the ViCom API and can therefore be used freely by you.

```
CViComGSMNWSInterfaceData::SDemodulationSettings cDemodSettings;
cDemodSettings.wFrequencyIndex = pcSCHInfoResult->wFrequencyIndex;
cDemodSettings.wBSIC = pcSCHInfoResult->wBSIC;
cDemodSettings.dwIndicatorOfSCHInfo = pcSCHInfoResult->dwIndicatorOfSCHInfo;

cDemodSettings.DemodulationRequestST1.eRequestType =
cDemodSettings.DemodulationRequestST2.eRequestType =
cDemodSettings.DemodulationRequestST4.eRequestType =
CViComGSMNWSInterfaceData::SDemodulationSettings::SDemodRequest::DEMODREQUEST_DO
NOTHING;
cDemodSettings.DemodulationRequestST3.eRequestType =
CViComGSMNWSInterfaceData::SDemodulationSettings::SDemodRequest::DEMODREQUEST_ST
ART_REQUEST;
cDemodSettings.DemodulationRequestST3.dwDemodulationTimeoutInMs = 10000;
cDemodSettings.DemodulationRequestST3.dwRequestIdent = 1;

pcGsmScanner->RequestDemodulationOfST1To4( cError, cDemodSettings );
```

The sample shown above demonstrates how to specifically request a System Information Type 3 message for a channel with a timeout of 10 seconds. When a requested message cannot be decoded, attempts are stopped after that time.

If a result can be demodulated, a structure is filled with the desired information. The most important information is stored in the hex dump field in the structure. It can be used to analyse it with a GSM Layer 3 compatible parser.

As a special case, some content of the System Information Type 3 message is decoded by the scanner itself to gain the current network infrastructure IDs, namely:

- CI
- LAC
- MCC
- MNC

These will be reported in the structure `SCellIdentResult`.



Please refer to the following Demodulation section of the GSM Network Scanner description to learn more about the new demodulation capabilities. The automatic demodulation described here is less flexible and complex than the new method.

6.1.2.5 Spectrum Measurement

The GSM network scanner also provides a way to create a spectrum of the GSM channels defined for the measurement. The spectrum is calculated from a set of raw measurements, which are the results of the FFTs applied to the digitized signal.

The data from the FFTs is reduced to a configurable number of spectrum power values for a GSM channel. Therefore, the channel bandwidth of 200 KHz is divided in N equally sized intervals. For each interval, the highest power value is used as characteristic value.

From that spectrum data one sub-spectrum result is derived. This time aggregation can either be done in the same way as described above, or the root-mean-square is calculated for the set of sub-channel peaks. To illustrate that calculation, the figure below shows the combination of 3 FFTs to one single sub-spectrum.

Each sub-spectrum is based on a number of FFTs. The number of FFTs is chosen to best-fit into the collection time constraint specified in the spectrum setting.

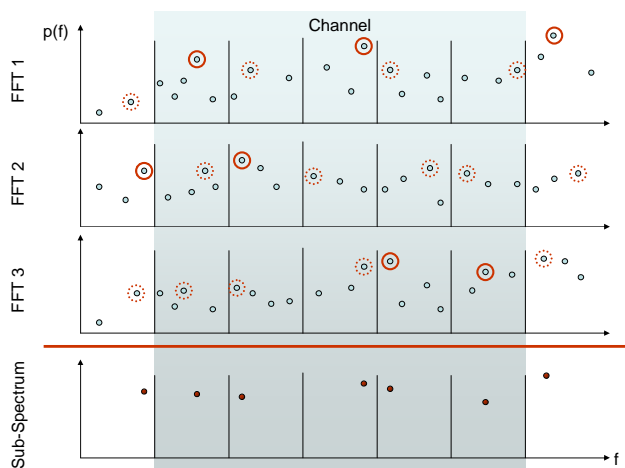


Figure 15 Spectrum Calculation

There are five sub-channels and three FFTs shown in the example. From the in this sample highly reduced amount of raw data, the sub-channel data per scan is found by using the peaks.

For time aggregation, also the peak power value is used: The first and the third sub-channel have their time peak in the first FFT. The second sub-channel is taken from the second one, and the last two values are derived from the third FFT.

The settings can be defined in the `CViComGSMNWSInterfaceData::SSpectrumSpec` structure when enabling the spectrum creation in the measurement details structure. The code fragment below shows how to enable the spectrum creation in the scanner (during an active measurement).

```
CViComGSMNWSInterfaceData::SMeasurementDetails cDetailSettings;
cDetailSettings.dwCount = 1;
cDetailSettings.pTableOfChannelMeasSpec = new
    CViComGSMNWSInterfaceData::SChannelMeasSpec[1];
cDetailSettings.pTableOfChannelMeasSpec[0].wFrequencyIndex = 0;
cDetailSettings.pTableOfChannelMeasSpec[0].bMEAS_SPECTRUM = TRUE;
cDetailSettings.SpectrumSpec.wCountOfPowerValuesPerChannel = 5;
cDetailSettings.SpectrumSpec.wCollectionTimeIn100us = 300; // == 30 msec
cDetailSettings.SpectrumSpec.eFreqDetector =
    CViComGSMNWSInterfaceData::SSpectrumSpec::FREQ_DETECTOR_PEAK;
cDetailSettings.SpectrumSpec.eTimeDetector =
    CViComGSMNWSInterfaceData::SSpectrumSpec::TIME_DETECTOR_PEAK;

EXEC_VICOM( "Requesting detailed measurements",
    pcGsmScanner->RedefineMeasurementDetails( cError, cDetailSettings ) );

delete [] cDetailSettings.pTableOfChannelMeasSpec;
```



Please note that the spectrum configuration can be changed during the measurement. Each time the spectrum configuration is changed, the new configuration is used for **all** spectrum measurements from then on. It is not possible to create different kinds of spectrum measurements for different frequencies in the same measurement at one time.

The results of such a request will be returned as part of the GSM NWS measurement result in the `CViComGSMNWSInterfaceData::SSpectrumResult` structure. The result contains a set of sub-spectrums. Such a sub-spectrum is the result of the combined FFTs of all measured channels, as described above.

Within the byte buffer, the data is organized primarily in sub-spectrums. In each sub-spectrum, the power values from each measured frequency are given in an ascending order. Data from one frequency is given in one block. This is depicted in the figure below. The term Δf refers to the fraction of one channel that is determined by the number of power values per channel.

Sub-Spectrum								
Channel 1			Channel 2			Channel 3		
Power	Power	Power	Power	Power	Power	Power	Power	Power
42	17	31	14	53	48	27	23	35
$f_{c1}-\Delta f$	f_{c1}	$f_{c1}+\Delta f$	$f_{c2}-\Delta f$	f_{c2}	$f_{c2}+\Delta f$	$f_{c3}-\Delta f$	f_{c3}	$f_{c3}+\Delta f$

Figure 16 Structure of Result Buffer

A sample shown below describes that structure and the power value conversion. In that picture, the buffer is shown in the upper part of the figure. Three sub spectrums were created in the sample for three frequencies (this implies there is only one power value per frequency).

In the byte buffer of the result structure, the list of spectrum power values is stored. Every power value result is specified in a 0.5 dB unit relative to on a minimum value, which is given in the field `SSpectrumResult::sMinPowerInDBm100`.

To get the real power values from the bytes in the buffer, the raw data has to be converted according to this formula:

$$dPower[i] = 100.0 * pcMeasResult->pSpectrumResult->sMinPowerValueInDBm100 + 0.5 * pcMeasResult->pSpectrumResult->pbBuffer[i];$$

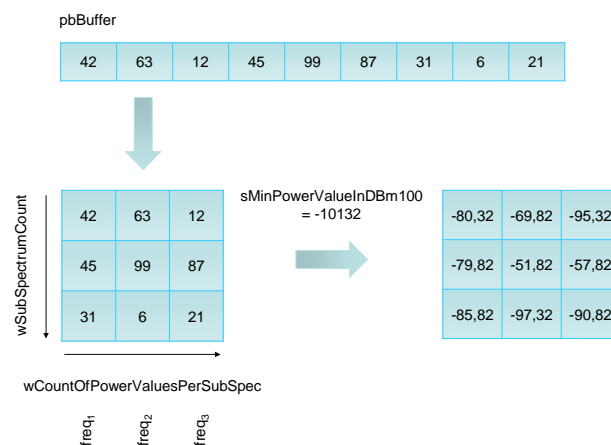


Figure 17 Spectrum Result Interpretation

The sub-spectrums contain the measured channels in the same order as they are given in the structure `ListExecutedMeasSpec`. That means, the first `wCountOfPowerValuesPerSubSpec` values in the buffer belong to the first entry in `ListExecutedMeasSpec`, the second set to the second entry and so on.

6.1.2.6 Channel Power Measurement

The power measurement mode that is enabled per default calculates one characteristic power value per every 50 msec (circa). Another mode to measure much more such power values is the channel power measurement mode. In this mode it is possible to get a very high resolution and therefore a big amount of data from the scanner.

A main concept in the channel power measurement mode is the reduction of the available signal bandwidth. Instead of using all power values from the 200 kHz bandwidth of a GSM channel, the GSM network scanner only uses the data from the 156 kHz interval around the center frequency.

This approach removes the influence of adjacent channels on the channel that is investigated. Since some of the signal power is therefore lost, the power is divided by 85%, because it has been shown that about 85% of the signal power of a GSM channel is within a 156 kHz bandwidth.

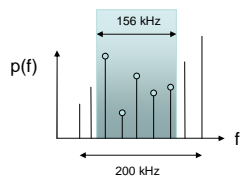


Figure 18 Reduction of used Signal Bandwidth

From the remaining input data, the FFT results are grouped to form one power value. One group consists of FFTs falling in a time-interval which is a multiple of 40 nsec. The set of power values calculated from the FFTs is then averaged by a root-mean-square algorithm to get the single power value of a sub-group.



Please note that the GSM network scanner does not synchronize on the time-slot to find the real timeslot borders in the channel power measurement mode. The term time-slot is used in this description and the interface documentation to clarify the calculation algorithm and the concept of dividing a time period of that length into sub-groups. Since the calculation of results is a continuous process and can be compared to a sliding window calculation, time-slot synchronization is not necessary.

For the requested number of result values per timeslot this calculation is repeated. The algorithm is illustrated in the figure depicted below.

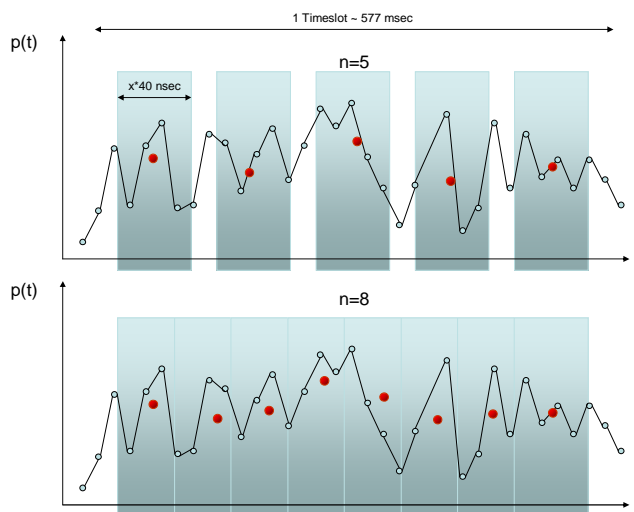


Figure 19 Channel Power Calculation Scheme

In the upper part of the sample shown above one time-slot is divided into 5 sub-groups. One interesting aspect shown is that it is not necessary that always the same amount of data is used in the average process. This can be seen in the first two groups, where four resp. five values contribute to the overall result.



Note that the data shown in the sample is nowhere near the amount of data used in real-world measurements. This is just another simplification.

It is also possible to create overlapping sub-groups. For example, if you use 500 times 40 nsec blocks for a subgroup and you divide the time-slot in 30 sub-groups, there will be an intersection of about 760 nsec between the groups. An example for this is shown in the lower part of the figure.

```
CViComGSMNWSInterfaceData::SMeasurementDetails cDetailSettings;
cDetailSettings.ChannelPowerSpec.wCountOfResultsPerGSMTTimeSlot = 30;
cDetailSettings.ChannelPowerSpec.wRMSLengthIn40ns = 500;
//...
```

The channel power measurement mode can be configured easily. There are only two attributes that must be specified: Firstly, the amount of time has to be set in which FFTs contribute to a power value. These must be done in multiples of 40 nsec. Secondly, the number of sub-groups within a time-slot must be given. In this way you can directly control the number of results, since for each measured channel exactly this number of results will be created.

In the API, the `SChannelPowerSpec` structure is used for that purpose and is part of the `SMeasurementDetails` structure. When the channel power measurement mode is enabled in the details structure, the `ChannelPowerSpec` field must be configured properly.

The result of the channel power measurements are similar to the ones created by the Spectrum mode. There is again an array of power values that follow some defined order. The actual power values in the buffer are specified in steps of 0.5 dB and have to be added to some given minimum power value. Inside the buffer, the values are structured similar in two dimensions: First they are grouped by channels, then by sub-group within one time-slot.

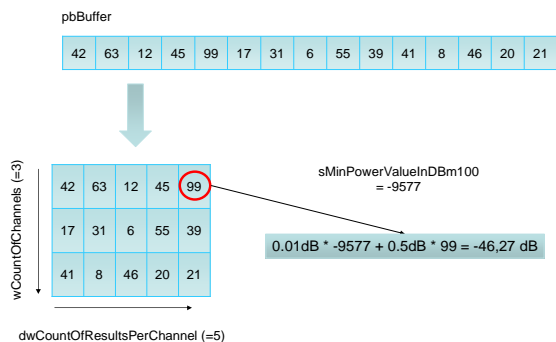


Figure 20 Result of Channel Power Measurement

The figure above shows the conversion from the linear result buffer into the aforementioned two dimensions and the calculation of the power values based on a specific sample. The calculation is done in exactly the same way as shown in the spectrum measurement mode.

That means: The i^{th} power value of the k^{th} channel can be calculated as follows:

$$P_k(i) = 0.01 * sMinPowerValueInDBm100 + 0.5 * pbBuffer[k * wCountOfResultsPerChannel + i];$$

6.1.2.7 Time-Slot Measurement

On one GSM channel, multiple BTS can send their information. In urban areas it might happen many times that for example ten different BTS transfer data on a physical channel. The signal from one such BTS can be separated from the others using the time-slot detection task. Therefore the TSMx software tries to detect as many different information streams on the signal and synchronize on the time-slots.

If the synchronization process is successful, the software is able to identify the Training Sequence Code. This TSC and the timing information are used to build a so-called cluster. In such a cluster data from one BTS is gathered (with a high certainty). To get to this point, several time slot measurements are combined. This is also necessary to reduce the presence of ghost codes.

The Time-Slot Measurement does not support any specialized configuration. You can simply enable or disable it in the measurement details structure and will receive results accordingly.

Results made in this mode are returned in two different lists. The first list contains the different clusters measured. Each cluster is made up a back reference to the frequency index, and the following attributes:

- The minimum power value `sMinPowerInDBm100` for a cluster and the total power range `wPowerRangeInDBm100`. In combination, both values specify the value range interval `[sMinPowerInDBm100; sMinPowerInDBm100 + wPowerRangeInDBm100]` in which the power values in the associated list are.
- The slot alignment of the timeslots (refer to the subsection Slot Alignment below for more information on that subject).
- The Training Sequence Code and the type of the time-slot. A time-slot may contain data from a normal or a dummy burst. Dummy bursts can be removed from the measurements by setting the `bMEAS_DB_REMOVAL` in the channel measurement specification.

The power values in the value list have to be set in relation to the interval specified in the cluster result. The calculation is depicted below and explained afterwards.

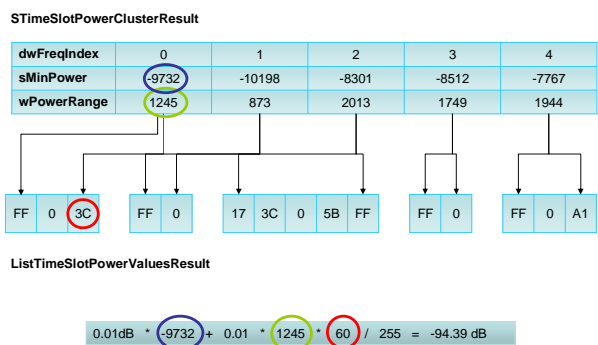


Figure 21 Result Structures of Time Slot Measurement Mode

From the `STimeSlotPowerClusterResult`, we know the minimum power value and the range. The time-slot power values are given in the value list as a fraction of the power range. This fraction is based on the maximum value of a byte, 255. So the complete calculation is as shown above:

- Multiply the power value `bPower` with the power range `wPowerRangeInDBm100` and divide the result by 255.
- When this intermediate result is multiplied with 0.01, the difference to the minimum value is available in dBm
- Now the minimum value `sMinPowerInDBm100` is transformed into dBm units (multiply with 0.01) and the difference calculated before is added.

The total time-slot power can be calculated in a similar way. To the result calculated above, the value from `bOffsetToTotalPowerInDBm10` can be added (after multiplying it by 0.1). The offset power specifies the total power of the signal (not only of the time-slot which is assigned to a BTS) as offset to the already available minimum value.

Slot Alignment

According to the GSM standard, one TDMA frame consists of eight timeslots. Each timeslot can contain at most 156.25 bits of information. Of course, the decimal part is not really a carrier of information, it is just a tribute to the timing constraints.

As an alternative to send the burst in that odd time delta, a GSM cell is allowed to insert a full filling bit after each 4th burst. The overall time of the TDMA frame is therefore not changed, but the timing properties of the burst have. With the knowledge whether the first or second option is used, additional information about the cell can be gained.

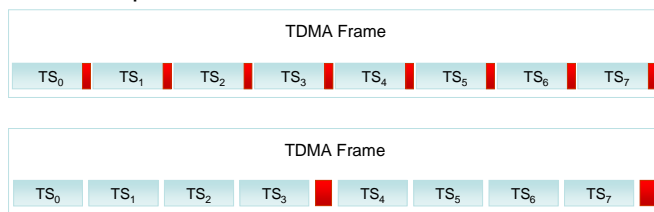


Figure 22 TDMA Slot Alignment Schemes in GSM

These aforementioned timing effects are used by the GSM NWS API to make an as-good-as-possible guess what slot alignment is used by a cell. The internal calculation of the slot alignment uses a probabilistic algorithm to finally find some probabilities for each of five possible positions (Time-Slots are equidistant, as shown in the figure above, upper half, or the fill bit is placed before every 0th, 1st, 2nd or 3rd timeslot.

The maximum probability is then referred to as 100%. Then the scanner software reports two results in form of a bitmask: One mask holds all that alignments that have a relative probability to the maximum of at least 95%. The second mask holds all that possibilities that have a probability that is at least 50% of the maximum peak. That behavior is shown below.

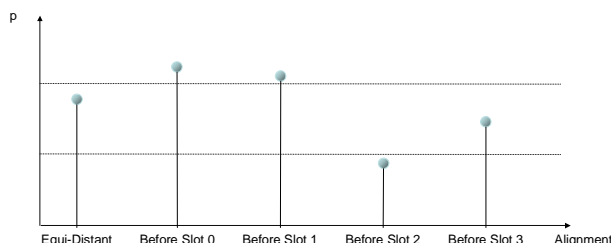


Figure 23 Slot Alignment Probabilities

The picture shows the situation, that it is most likely a fill-bit is used and it is placed directly before the first scan.



The more different time-slots could be measured and the more these time-slots have different positions to each other, the better the results will be and the more exactly can the scanner report the probabilities.

6.1.2.8 Removing Dummy Burst Measurements

Dummy Bursts can have a significant influence on the results of other measurement types. It is possible to remove those dummy bursts after demodulation and before other measurement task perform their calculation, which is especially useful for the channel power and the time slot measurements.

However, the algorithm applied to remove those dummy bursts is a very time consuming task and consumes some of the PCs computing time. It might therefore be necessary to selectively enable and disable that removal for specific channels during the measurement. To achieve this, it is only necessary to set the flag `bMEAS_DB_REMOVAL` in the structure `SChannelMeasSpec` accordingly.

6.2 The GSM Network Scanner Demo Application

Like with all other ViCom APIs, a demo application is packaged with the GSM network scanner. The interface is similar to the other demo applications. The application window is a dialog that is organized in columns.

The left-most column is similar to the one from other demo applications. It contains the control center in which buttons are provided that relate to the methods in the `CViComGSMNWSInterface` class (and their base interface). The order of the buttons implies the logical order in which they should be executed.

The middle column provides fields for setting the measurement and technology specific properties. In the frequency list, the channels that shall be measured must be specified. This is the only setting that must be done before starting the measurement. For each frequency, the automatic demodulation of system type messages can be specified upfront, as well as the frequency index (which must be ascending, starting by 0 and unique).

The right column contains a mixture of functions, like settings management, basic settings and result evaluation (i.e. showing the results which are written into a text-file in an editor).

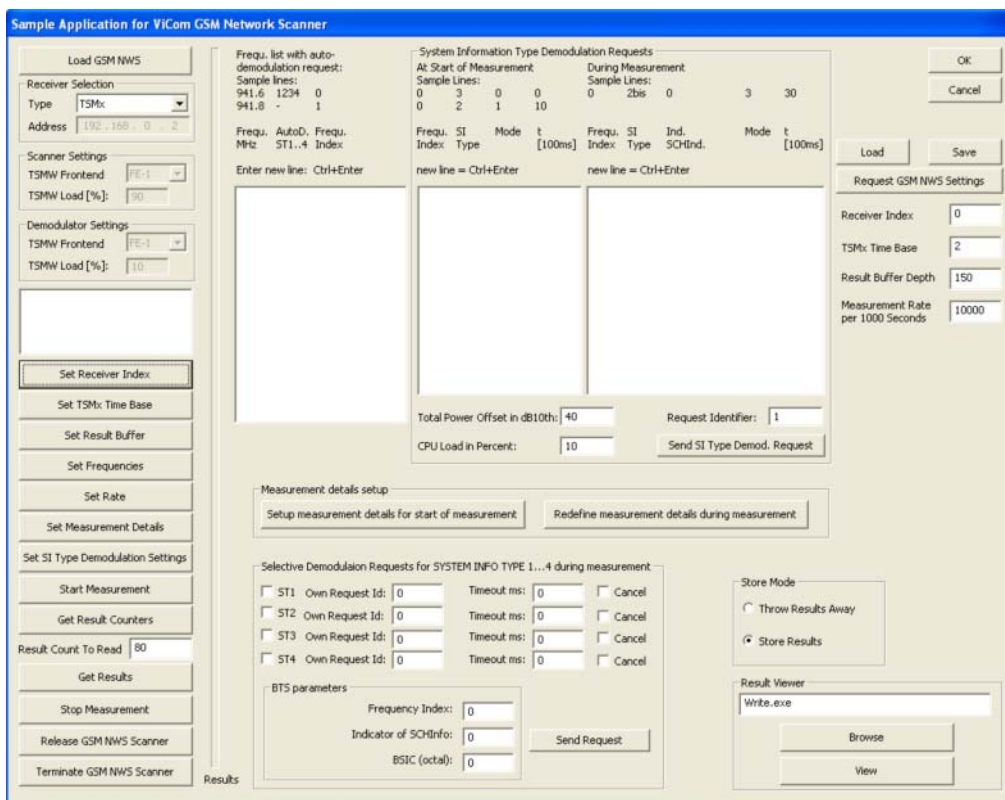


Figure 24 GSM Network Scanner Demo Application

6.2.1 Basic Channel Configuration

The first step when using the application is to define the list of frequencies that shall be scanned. Therefore, the following steps must be done:

- Load the TSMx device (by pressing “Load GSM NWS”) and set the initial values for Time Base, Result Buffer and Measurement Rate by pressing the related buttons. This is not really necessary when the application is started for the first time or the value has not been changed, but once a setting has been modified it is required to update the parameters in the TSMx device.
- Put the list of frequencies into the list in the middle column. You don’t have to enter the auto-demodulation settings and frequency index; they are added automatically with default values. A line in the edit control contains up to three elements, each separated by a whitespace:
 - Channel center frequency in MHz (decimals follow a period)
 - Auto-Demodulation settings for System Type Information messages. You must specify which shall be enabled by entering the numbers in one block. For example, if the SYSTPE 1 and 4 shall be decoded, enter 14.
 - The frequency index that shall be used later in the result. This is not mandatory and will be filled automatically when you enter the frequencies.



Note that you can enter new lines by pressing Ctrl+Enter when the focus is in the list control.

- Press “Set Frequencies” button to transfer the settings to the TSMx device
- Optionally, a more detailed setup of the measurement can be made. Therefore, press the “Setup measurement details for start of measurement” button below the frequency list. See the section below for a more detailed discussion.
Per default, only SCH measurements are enabled.
- By pressing “Start Measurement”, the measurement can be started now.
- To fetch some results, check the number of result available and transfer some of them to the PC (using the buttons “Get Result Counter” and “Get Results”).

6.2.2 Specifying Measurement Details

Most of the settings necessary to define the different measurement tasks which have been explained in the previous chapter must be setup in a special dialog that is shown below. The same dialog is used to initialize the measurement details before a measurement is started and when measurement configuration is changed while the TSMx is active. Slight differences concerning the buttons at the bottom of the dialog can be seen when comparing the dialogs, but the major part of the dialog is consistent.

To show the dialog, either press “Setup measurement details for start of measurement” before you pressed the “Start Measurement” button. Once the measurement is active, the button “Redefine measurement details during measurement” can be used.

The configuration should be straight-forward after the previous chapter has been read. The input fields reflect the fields in the C-structures for many parts of the `SMeasurementDetails`. Besides the settings for the spectrum and channel power measurement modes, the dialog also provides a way to enable and disable the measurement tasks.

One important aspect is that the configuration can be applied to a different scope of channels. It's possible to modify the details for a single channel, for all channels or to modify the spectrum and channel power measurement modes without changing other settings. The behaviour can be defined in the lower part using the different radio buttons.

Figure 25 Measurement Details Dialog

Even a subset of the measurement frequencies can be configured. Click on the button “Press button for individual setup” that appears when the radio button “Individual setup for a list of frequencies” is selected. In the upcoming dialog, the active measurement tasks can easily be defined per each configured frequency.



The spectrum and channel power measurement settings cannot be defined on a per-channel basis, as the dialog illustrates. Both configurations are valid for all channels that perform that measurement tasks. Each modification always concerns all active measurements.

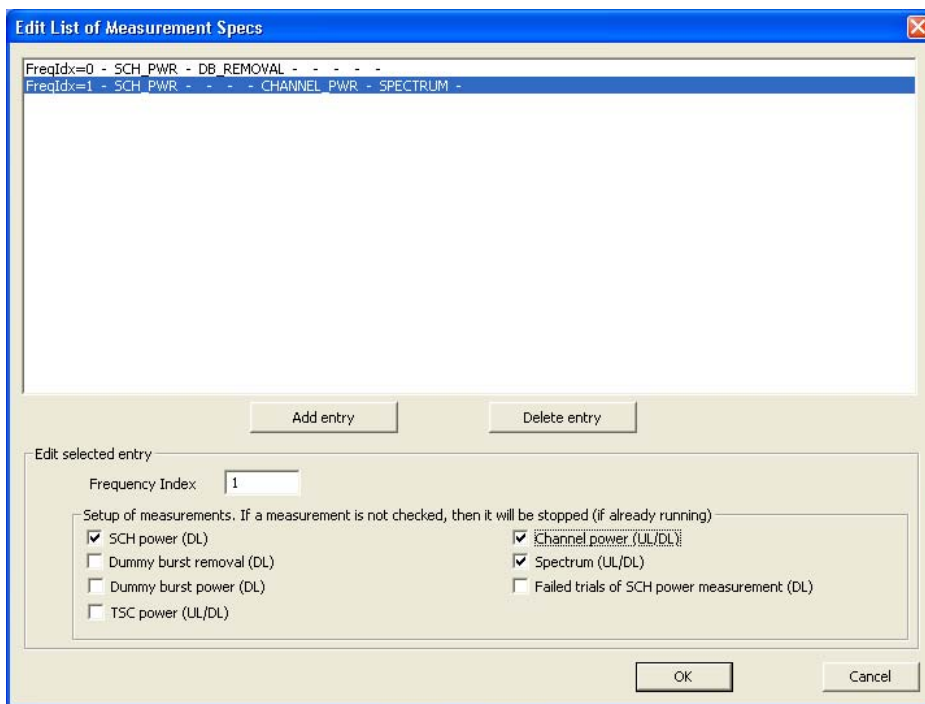


Figure 26 Measurement Mode Specification

6.2.3 Requesting System Type Information

Once a measurement has been started, you can request the System Type Information 1-4 for a specific BTS. Therefore, the lower part of the main dialog offers the appropriate controls.

But before such a request can be sent, it is required that the SCH has been demodulated successfully. This is due to the fact that the SCH contains information required to synchronize and demodulate the BCCH (namely the BSIC).

For each message type, a user-managed request id can be specified. This helps identifying to which request a result belongs. The dialog also allows setting a timeout, after which the demodulation request is stopped. A currently active demodulation request can also be cancelled again. The related checkbox has to be selected when the request is sent.

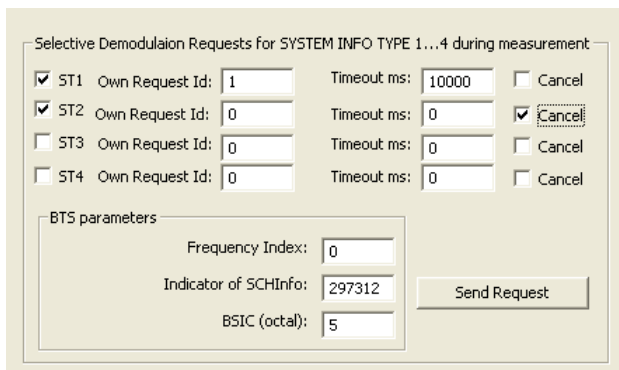


Figure 27 System Type Information Demodulation Options

For all requests, the channel (identified by its center frequency index), the indicator of the SCH Info (taken from the SCH demodulation result) and the BTSs BSIC have to be given to the network scanner.

6.3 The Sample

It's time to get our hands on some code, and this section will go straight to the technical stuff. Since a lot of different measurement types can be done with GSM Network Scanner API, the sample application has not one single task. We define several goals that the sample shall fulfil:

- Task 1: Always the show the strongest channel found so far. This means that we always want to see the highest power value and the channel for which we detected that value.
- Task 2: Measure a 2 MHz broad spectrum around all strongest channels that have been found so far.
- Task 3: Decode the SIT 3 and show the network identifiers contained there for the strongest channel.

To avoid confusion and to keep the sample as small as possible, many convenience functions are not implemented. The samples job is simply to show how to use the API.

6.3.1.1 Code Listing

```

#include "stdafx.h"

#include <windows.h>
#include <iostream>
5  #include <conio.h>

#include "ViComBasicInterfaceData.h"
#include "ViComError.h"
#include "ViComLoader.h"
10 #include "ViComGSMNWSErrors.h"
#include "ViComGSMNWSInterface.h"
#include "ViComGSMNWSInterfaceData.h"

using namespace std;
15

/*****

#define EXEC_VICOM( msg, call ) \
20   if ( strlen(msg) ) { cout << msg << "..."; } \
   call; \
   if ( cError.GetErrorCode() > 0 ) { \
   char* szErr = new char[strlen(cError.GetErrorString()) + 1]; \
   strcpy(szErr, cError.GetErrorString()); \
25   throw CViComError(cError.GetErrorCode(), szErr); } \
   if ( strlen(msg) ) { cout << "ok" << endl; }

*****/

void RunGsmNetworkScannerSample()
30 {
   CViComError cError;
   CViComLoader< CViComGSMNWSInterface > cGsmNwsLoader;

   EXEC_VICOM( "Loading GSM Network Scanner",
35     cGsmNwsLoader.LoadTsmx( cError ) );

   EXEC_VICOM( "Retrieving special interface",
     CViComGSMNWSInterface* pcGsmScanner = cGsmNwsLoader.GetpInterface( cError ) );

```

```

40     //
    // Now we create a list of all P-GSM downlink channels in germany.
    // There are 124 such channels, ranging from 935 MHz to 960 MHz.
    // Each channel has a bandwidth of 200 KHz.
    //
45     // It is IMPORTANT to have the values in the table sorted in
    // ascending order.
    //
    int nChannelCount = 124;
    CViComGSMNWSInterfaceData::SFrequencySetting* ptFrequencySettings =
50     new CViComGSMNWSInterfaceData::SFrequencySetting[ nChannelCount ];
    for ( int i=0; i<nChannelCount; i++ )
    {
        ptFrequencySettings[ i ].dCenterFrequencyInHz = 935200000 + i * 200000;
65     ptFrequencySettings[ i ].bRequestAutoDemodulationOfST1 = false;
70     ptFrequencySettings[ i ].bRequestAutoDemodulationOfST2 = false;
75     ptFrequencySettings[ i ].bRequestAutoDemodulationOfST3 = false;
        ptFrequencySettings[ i ].bRequestAutoDemodulationOfST4 = false;
    }

    CViComGSMNWSInterfaceData::SChannelSettings cChannelSettings;
    cChannelSettings.dwCount = nChannelCount;
    cChannelSettings.pTableOfFrequencySetting = ptFrequencySettings;
    pcGsmScanner->SetFrequencyTable( cError, cChannelSettings );

    CViComGSMNWSInterfaceData::SMeasurementRate cMeasRate;
    cMeasRate.dwValuePer1000Sec = 10000; // measure every 100 milliseconds
    EXEC_VICOM( "Setting measurement rate to 100 millisecond",
        pcGsmScanner->SetMeasurementRate( cError, cMeasRate ) );

    EXEC_VICOM( "Start measurement",
        pcGsmScanner->GetBasicInterface().StartMeasurement( cError ) );

    // Required to find the strongest channel
    short nMaxPowerValue = -10000;
    double nMaxPowerFrequency = 0;

    while ( true )
    {
80     // Check for keyboard stroke to exit demo
        if ( _kbhit() )
        {
            break;
        }

85     CViComGSMNWSInterfaceData::SMeasResult* pcMeasResult;

        CViComBasicInterface::SResultCounters* pcResultCounters;
        EXEC_VICOM( "", pcResultCounters =
90         pcGsmScanner->GetBasicInterface().GetResultCounters( cError ) );

        if ( pcResultCounters->dwCountOfBufferdResults == 0 )
        {
            Sleep( 100 );
            continue;
95     }

        EXEC_VICOM( "",
            pcMeasResult = pcGsmScanner->GetResult( cError, 100 ) );

100     if ( ! pcMeasResult )
        {
            continue;
        }

105     // Task 1: Find strongest frequency
    //
    // From the "normal" list of channel power values, we try to find the
    // strongest channel. This is a repetitive process: Each time a new
    // set of measurement values arrives, the list is searched for a more

```



```

110         // powerful channel. If one is found, it is reported.
           //
           // Also, for each channel further measurements are enabled.

SViComList<CViComGSMNWSInterfaceData::SMeasResult::SPowerResult>::SLinkedObject*
115 pcPowerResult
    = pcMeasResult->ListPowerResults.pFirst;
    while ( pcPowerResult )
    {
120         if ( pcPowerResult->sPowerInDBm100 > nMaxPowerValue )
            {
                nMaxPowerValue = pcPowerResult->sPowerInDBm100;
                nMaxPowerFrequency = ptFrequencySettings[ pcPowerResult-
>wFrequencyIndex ].dCenterFrequencyInHz;

125                 // Power values are returned as integer (exactly, short)
                // values. To include decimals, the floating value is
                // multiplied by 100.
                cout << "New max value: " << ( nMaxPowerValue / 100.0 );

130                 // The frequency is returned as index from the channel settings
                // list.
                cout << " found on freq " << ( nMaxPowerFrequency / 1000000 );
                cout << " MHz " << endl;

135                 CViComGSMNWSInterfaceData::SMeasurementDetails cDetailSettings;
                cDetailSettings.dwCount = 1;
                cDetailSettings.pTableOfChannelMeasSpec = new
CViComGSMNWSInterfaceData::SChannelMeasSpec[1];
140                 cDetailSettings.pTableOfChannelMeasSpec[0].wFrequencyIndex =
pcPowerResult->wFrequencyIndex;
                cDetailSettings.pTableOfChannelMeasSpec[0].bMEAS_SPECTRUM = TRUE;
                cDetailSettings.SpectrumSpec.wCountOfPowerValuesPerChannel = 5;
                cDetailSettings.SpectrumSpec.wCollectionTimeIn100us = 300; // == 30
msec

145                 cDetailSettings.SpectrumSpec.eFreqDetector =
CViComGSMNWSInterfaceData::SSpectrumSpec::FREQ_DETECTOR_PEAK;
                cDetailSettings.SpectrumSpec.eTimeDetector =
CViComGSMNWSInterfaceData::SSpectrumSpec::TIME_DETECTOR_PEAK;
                cDetailSettings.ChannelPowerSpec.wCountOfResultsPerGSMTTimeSlot = 10;
150                 cDetailSettings.ChannelPowerSpec.wRMSLengthIn40ns = 500;

                EXEC_VICOM( "Requesting detailed measurements",
                    pcGsmScanner->RedefineMeasurementDetails( cError, cDetailSettings
) );

155                 delete [] cDetailSettings.pTableOfChannelMeasSpec;
            }

pcPowerResult = pcPowerResult->pNext;
160     }

    // Task 2: Show the spectrum result. Therefore, we reduce the number of
    // power values
    // to the maximum peak and average value in each sub-spectrum channel.
165     if ( pcMeasResult->pSpectrumResult )
        {
            CViComGSMNWSInterfaceData::SMeasResult::SSpectrumResult* pSpectrum =
pcMeasResult->pSpectrumResult;

170             // Calculate the frequency difference between power values in the
            // result buffer
            double nFrequencySpacingInChannel
                = 200000.0 / pSpectrum->wCountOfPowerValuesPerChannel;

175             // Iterating over the buffer, splitting it as described in the manual
            // into sub-spectrums, channels and power sub-channels.
            DWORD dwChannelCount =
                pSpectrum->wCountOfPowerValuesPerSubSpec
                / pSpectrum->wCountOfPowerValuesPerChannel;
180

```

```

byte* pvbCurrentPowerValue = pSpectrum->pbBuffer;
for ( WORD nSpectrum = 0;
      nSpectrum < pSpectrum->wSubSpectrumCount; nSpectrum++ )
185   {
      cout << "Sub-Spectrum " << nSpectrum << ":" << endl;

SViComList<CViComGSMNWSInterfaceData::SChannelMeasSpec>::SLinkedObject*
      pCurrentMeasSpec = pcMeasResult->ListExecutedMeasSpec.pFirst;
190   for ( WORD nChannel = 0; nChannel < dwChannelCount; nChannel++ )
      {
          double nAverage = 0;
          byte nMaxPowerValue = 0; // relative to sMinPowerValueInDBm100;
          double nMaxFrequency = 0;
195
          WORD nChannelIndex = pCurrentMeasSpec->wFrequencyIndex;
          for ( WORD nValueIdx = 0;
                nValueIdx < pSpectrum->wCountOfPowerValuesPerChannel;
200   nValueIdx++ )
          {
              if ( *pvbCurrentPowerValue > nMaxPowerValue )
              {
                  nMaxPowerValue = *pvbCurrentPowerValue;
205
                  // Calculation of the current frequency is a little
                  // complication. From the center frequency of the channel,
                  // 100000 Hz are subtracted to find the lower border of the
                  // bandwidth. Then, the nValueIdx-th frequency is found by
                  // adding the frequency delta nValueIdx times and a half to
210   // find the center of the sub-channel. Finally the unit is
                  // converted to MHz.
                  nMaxFrequency =
                    ( ptFrequencySettings[ nChannelIndex
215   ].dCenterFrequencyInHz
                      - 100000
                      + nValueIdx * nFrequencySpacingInChannel
                      + nFrequencySpacingInChannel / 2) / 1000000;
                }

220   nAverage += *pvbCurrentPowerValue;

                pvbCurrentPowerValue++;
            }

225   double dAverage = 0.01 * pSpectrum->sMinPowerValueInDBm100
            + 0.5 * (nAverage / pSpectrum->wCountOfPowerValuesPerChannel);
            double dMaxValue = 0.01 * pSpectrum->sMinPowerValueInDBm100
            + 0.5 * nMaxPowerValue;

230   cout << "\t Ch " << nChannelIndex << " Avg: " << dAverage;
            cout << " Max: Freq. " << nMaxFrequency;
            cout << " Pow: " << dMaxValue << endl;

235   pCurrentMeasSpec = pCurrentMeasSpec->pNext;
        }
    }

    // Task 3: Issue a request for the demodulation of SIB 3 if SCH Result
    // is available for a channel. The result if any will be shown below.
240   SViComList<CViComGSMNWSInterfaceData::SMeasResult::SSCHInfoResult>::SLinkedObjec
    t* pcSCHInfoResult
        = pcMeasResult->ListSCHInfoResults.pFirst;
        while ( pcSCHInfoResult )
245   {
            CViComGSMNWSInterfaceData::SDemodulationSettings cDemodSettings;
            cDemodSettings.wFrequencyIndex = pcSCHInfoResult->wFrequencyIndex;
            cDemodSettings.wBSIC = pcSCHInfoResult->wBSIC;
            cDemodSettings.dwIndicatorOfSCHInfo = pcSCHInfoResult-
250   >dwIndicatorOfSCHInfo;
        }
    }
}

```

```

        cDemodSettings.DemodulationRequestST1.eRequestType =
CViComGSMNWSInterfaceData::SDemodulationSettings::SDemodRequest::DEMODREQUEST_DO
NOTHING;
255     cDemodSettings.DemodulationRequestST2.eRequestType =
CViComGSMNWSInterfaceData::SDemodulationSettings::SDemodRequest::DEMODREQUEST_DO
NOTHING;
        cDemodSettings.DemodulationRequestST3.eRequestType =
CViComGSMNWSInterfaceData::SDemodulationSettings::SDemodRequest::DEMODREQUEST_ST
260     ART_REQUEST;
        cDemodSettings.DemodulationRequestST3.dwDemodulationTimeoutInMs = -1;
        cDemodSettings.DemodulationRequestST3.dwRequestId = 3;
        cDemodSettings.DemodulationRequestST4.eRequestType =
CViComGSMNWSInterfaceData::SDemodulationSettings::SDemodRequest::DEMODREQUEST_DO
265     NOTHING;

        EXEC_VICOM( "Requesting ST Demodulation ",
                    pcGsmScanner->RequestDemodulationOfST1To4( cError, cDemodSettings )
                );
270     pcSCHInfoResult = pcSCHInfoResult->pNext;
    }

    // If available, we show the decoded information from the system type 3
    // message which contains the GCI of a BTS.
275     SViComList<CViComGSMNWSInterfaceData::SMeasResult::SCellIdentResult>::SLinkedObj
ect* pcCellIdResult
        = pcMeasResult->ListCellIdentResults.pFirst;
        while ( pcCellIdResult )
280     {
            cout << "Decoded ST 3 for Freq ";
            cout << ( ptFrequencySettings[
                pcCellIdResult->wFrequencyIndex ].dCenterFrequencyInHz ) /
(1000*1000);
285             cout << ": CI " << pcCellIdResult->wCI;
                cout << " LAC " << pcCellIdResult->wLAC;
                cout << " MCC " << pcCellIdResult->wMCC;
                cout << " MNC " << pcCellIdResult->wMNC << endl;

                pcCellIdResult = pcCellIdResult->pNext;
290         }
    }

    EXEC_VICOM( "Stop measurement",
                pcGsmScanner->GetBasicInterface().StopMeasurement( cError ) );
295

    EXEC_VICOM( "Waiting for stop measurement signal",
                pcGsmScanner->GetBasicInterface().HasMeasurementStopped( cError, 1000 )
    );
300

    EXEC_VICOM( "Unloading GSM NWS Scanner...",
                cGsmNwsLoader.ReleaseTsmx( cError, false ) );
    }

    /*****
305
int _tmain(int argc, _TCHAR* argv[])
{
    try
310     {
        cout << "ViCom GSM Network Scanner Demo" << endl << endl << endl;

        RunGsmNetworkScannerSample();
    }
    catch ( CViComError& cError )
315     {
        cout << cError.GetErrorString() << endl;
        return cError.GetErrorCode();
    }
320     return 0;
}

```

6.3.1.2 Code Structure

The code sample contains one central method that holds all the program logic. The other parts of the listing (header and footer) prepare the measurement resp. to the formal stuff required to run code using the ViCom API.

First thing to do is to load and initialize the network scanner, which can be seen in lines 31-68. The thing to remark here is the way how frequencies are specified: The list of frequencies is created dynamically and holds all 124 GSM-900 channels. In the next lines, some other standard settings are defined, after which the measurement is started.

The central element of the sample is an endless while-loop that can be left only by hitting some key on the keyboard or by an exception. In that loop, the network scanner is requested to return some measurement result structure. If none is available, there is a short sleep phase and the buffer is checked again.

Per default, only the default power measurement is enabled. When a power measurement result is found and it is the maximum value found so far, further measurement modes are enabled. This happens in the lines 135-152. For the frequency the new maximum has been measured for is used to request spectrum measurements and SCH Info demodulation for.

The spectrum result is processed next. In lines 163-232 the code is shown to do a complete scan of the spectrum results. It is divided in sub-spectrum (first loop), channels (second loop) and a list of power values per channel (third, inner-most loop). In that inner-most loop, the maximum of the power values of one channel is searched and displayed.



Besides the maximum, an averaged value is returned. The calculation shown here is NOT a meaningful average calculation, but it is used here to simplify the code. If you really want to average such values, the power values must be transformed into a linear scale first (and vice-versa later).

Calculation of the frequency is a complicated task, as can be seen in lines 208-212. The center frequency of the currently investigated channel must be derived from the list of executed measurements. Therefore this linked list is iterated parallel to the iteration over the channels in the sub-spectrums.

Since there is a predefined number of power values per channel, the bandwidth is split into equally sized sub-channel intervals (see line 170). The index of the power value gives the distance from the lower border (which is found by subtracting 100 kHz from the center frequency) of the channel in such intervals. To get the center frequency of the sub-channel, half of sub-channel width is added finally.

Starting in line 220, the transformation from the decoded power values into a floating point representation is shown.

The third task needs some additional interaction with the network scanner. In the lines 236-264, the SCH demodulation result is processed. From that result, the BSIC and some other indicators are used to send a specific demodulation request to the scanner for that BTS (line 261).



An important thing to notice is that the timeout is put as -1 in the sample to let the network scanner try as long as possible to decode a CI result. You'll have to find practical values here for the network, if you want to have the time-out feature enabled (which can be circumvented by cancelling the request manually, for instance when a cell has been found or some other special condition is met).

This request will be answered with a CellId result, if the demodulation can be done successfully. Such a 0result processing is shown in lines 268-283.

6.4 GSM BCH Demodulation

The GSM network scanner has also capabilities to demodulate system type information sent on the BCH. Therefore the ViCom provides an API to the demodulation functionality as described in the general section. To show the usage of the API, the demo application has been enhanced, as shown in the screenshot below.

Following the description how to use the API a code listing completes the introduction into the usage of the GSM NWS API, as it is the case in the other chapters.



The measurement results from the GSM network scanner demodulation process are derived using data from different time slots. This is reflected in the start and stop time arguments of the demodulation result structure.

6.4.1 Sample Application

The sample application looks different in the new ViCom delivery. In the middle column, two new lists can be used to enter the data requests for demodulation. These lists are marked with a red box in the screenshot shown below.

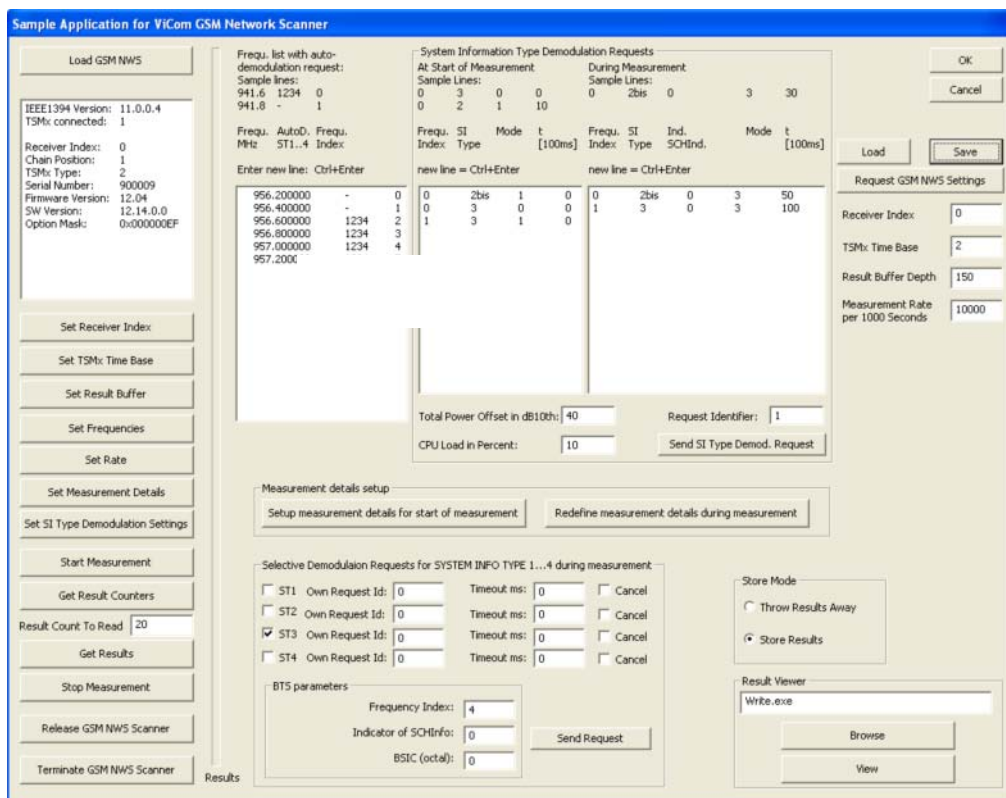


Figure 28 Enhanced Sample Application GSM Demodulator

In the left list, the configuration is entered that is used when the measurement is started to setup an initial configuration of the demodulation process. Four values have to be entered in one line, and each line is one configuration record. These values have the following meaning:

- Channel Index – the index of the frequency entered in the list left for which the configuration records shall be applied
- System Information Type – The type id of the message that shall be demodulated. This is not necessarily a number, valid values are also 2bis, 2ter etc. besides the numbered messages.
- Demodulation Type – Can be either 0, 1 or 2. 0 means that the demodulation for the system information shall be done until it has been performed successfully for one time. 1 should be used when the demodulation shall be requested dynamically (using the right list). 2 means that the demodulation shall be performed in regular intervals. This value corresponds to the values defines in `etSIType_DEMOD_MODE`.
- The last value is only used when the third value was set to 2. This value defines the delay that the demodulator delays the next demodulation attempt once the message has been decoded successfully.

There can be any number of such rows in the list. When pressing the “Set SI Type Demodulation Setting” button, these values are used to configure the scanner. Besides the configuration defined in the list, the two values entered in the text fields below are transferred to the scanner.



Please note that it you can NOT use the SI Type demodulation together with the automatic demodulation of System Type Information 1 to 4. The new demodulation algorithm can fully replace the formerly used configuration and provides better performance and more configuration options. Make sure to fully migrate existing software when using the new modes.

The list on the right side is used to define the demodulation tasks during the measurement. The content can be defined upfront, but pressing the button “Send SI Type Demod Request” is only supported when a measurement is active.

The main differences to the left list are listed below:

- The records are defined related to a specific basestation. The third column in the list should be an entry formerly measured with the scanner and can be taken from the standard measurement results (the internal basestation id that is assigned by the scanner must be used here).
- The demodulation types are now restricted to values 3-8, which correlate to the remaining definitions from the `etSIType_DEMOD_MODE` enumeration.

Additionally, the request identifier can be specified to distinguish different requests when the results are processed, since they may occur in any order.

6.4.2 Code Listing

Equivalent to the former chapters, below a small sample shows how to use the API in your programs. The program below simply starts demodulation of the System Information Type 3 message on all 124 channels in the GSM 900 band. Once a message has been demodulated, the sample starts a request to demodulate the System Information Type 13 message that is available in GPRS supporting networks only.

```

#include "stdafx.h"

#include <conio.h>
#include <iostream>
5  #include <windows.h>

#include "ViComBasicInterface.h"
#include "ViComError.h"
#include "ViComLoader.h"
10 #include "ViComGSMNWSErrors.h"
#include "ViComGSMNWSInterface.h"
#include "ViComGSMNWSInterfaceData.h"

using namespace std;

15
/*****

#define EXEC_VICOM( msg, call ) \
20   cout << msg << "..."; \
   call; \
   if ( cError.GetErrorCode() > 0 ) { \
       char* szErr = new char[strlen(cError.GetErrorString()) + 1]; \
       strcpy(szErr,cError.GetErrorString()); \
       throw CViComError(cError.GetErrorCode(), szErr); } \
25   cout << "ok" << endl;

*****/

```

```

30 void RunBchDemodulationSample()
   {
       CViComError cError;
       CViComLoader< CViComGSMNWSInterface > cBchDemodulator;

35         //
           // Below is the standard initialization sequence of a ViCom module.
           //
       EXEC_VICOM( "Loading GSM Scanner",
40           cBchDemodulator.LoadTsmx( cError ) );

       EXEC_VICOM( "Retrieving special interface",
           CViComGSMNWSInterface* pcNws = cBchDemodulator.GetpInterface(
               cError ) );

45         //
           // Common initialization sequence
           //
       EXEC_VICOM( "Setting receiver index",
50           pcNws->GetBasicInterface().SelectReceiver( cError, 0 ) );

       EXEC_VICOM( "Setting time base",
           pcNws->GetBasicInterface().SetTimebaseSynchronisationMode( cError,
               CViComBasicInterfaceData::TIME_BASE_SYNC_MODE_PPS_WATCHED_BY_GSM ) );

55         CViComBasicInterfaceData::SResultBufferDepth cResultBufferDepth;
           cResultBufferDepth.dwValue =
       CViComBasicInterfaceData::SResultBufferDepth::dwMax;
           EXEC_VICOM( "Setting result buffer",
60           pcNws->GetBasicInterface().SetResultBufferDepth( cError,
               cResultBufferDepth ) );

           //
           // The measurement rate and mode must be specified to produce a
           // meaningful measurement result.
65           // Set measurement mode to the smallest common divisor, which is HIGH SPEED
           //
       CViComGSMNWSInterfaceData::SMeasurementRate cMeasRate;
           cMeasRate.dwValuePer1000Sec =
       CViComGSMNWSInterfaceData::SMeasurementRate::dwMaxPer1000SecForTSMU;
70           EXEC_VICOM( "Setting Measurement Details",
               pcNws->SetMeasurementRate( cError, cMeasRate ) );

           //
           // We try to demodulate data from the complete GSM 900 band.
           // This is just a test case to demonstrate the usage of the API.
           //
75           const DWORD FREQ_COUNT = 124;

           CViComGSMNWSInterfaceData::SchannelMeasSpec MEASSPECS[ FREQ_COUNT ];

80           static CViComGSMNWSInterfaceData::SFrequencySetting
               FREQUENCIES_IN_MHZ[ FREQ_COUNT ];
           for ( unsigned int i = 0; i < FREQ_COUNT; i++ )
           {
85               FREQUENCIES_IN_MHZ[ i ].dCenterFrequencyInHz = ( 935.2 + 0.2 * i ) * 1000
                   * 1000;

                   //
                   // It is important to set these values to false when the
90                   // new demodulator is used.
                   //
               FREQUENCIES_IN_MHZ[ i ].bRequestAutoDemodulationOfST1 = false;
               FREQUENCIES_IN_MHZ[ i ].bRequestAutoDemodulationOfST2 = false;
               FREQUENCIES_IN_MHZ[ i ].bRequestAutoDemodulationOfST3 = false;
95               FREQUENCIES_IN_MHZ[ i ].bRequestAutoDemodulationOfST4 = false;

               MEASSPECS[ i ].wFrequencyIndex = i;
               MEASSPECS[ i ].bMEAS_SCH = TRUE;
               MEASSPECS[ i ].bMEAS_CHANNEL_POWER = FALSE;

```



```

100     MEASSPECS[ i ].bMEAS_DB_POWER = FALSE;
        MEASSPECS[ i ].bMEAS_DB_REMOVAL = FALSE;
        MEASSPECS[ i ].bMEAS_REPORT_FAILED_TRIALS = FALSE;
        MEASSPECS[ i ].bMEAS_SPECTRUM = FALSE;
105     MEASSPECS[ i ].bMEAS_TSC = FALSE;
    }

    CViComGSMNWSInterfaceData::SChannelSettings cFreqSettings;
    cFreqSettings.dwCount = FREQ_COUNT;
    cFreqSettings.pTableOfFrequencySetting = FREQUENCIES_IN_MHZ;
110     EXEC_VICOM( "Setting Frequency Table",
        pcNws->SetFrequencyTable( cError, cFreqSettings ) );

    //
115     // Now the demodulation details can be defined. In this sample, different
    // decoding strategies are shown that are supported by the ViCom demodulator.
    //
    CViComGSMNWSInterfaceData::S_SIType_Requests::S_SIType_Request
    STIREQUESTS[FREQ_COUNT * 2];
120     for ( unsigned int i = 0; i < FREQ_COUNT; i++ )
        {
            STIREQUESTS[2 * i].dwChannelIndex = i;
            STIREQUESTS[2 * i].dwSCHIndicator = 0;
            STIREQUESTS[2 * i].eDemodulationMode =
125     CViComGSMNWSInterfaceData::SITYPE_DEMOD_ONCE;
            STIREQUESTS[2 * i].eSIType = CViComGSMNWSInterfaceData::SITYPE_3;
            STIREQUESTS[2 * i].wRepetitionDelayIn100ms = 0;

            STIREQUESTS[2 * i + 1].dwChannelIndex = i;
            STIREQUESTS[2 * i + 1].dwSCHIndicator = 0;
            STIREQUESTS[2 * i + 1].eDemodulationMode =
130     CViComGSMNWSInterfaceData::SITYPE_DEMOD_ON_CMD;
            STIREQUESTS[2 * i + 1].eSIType = CViComGSMNWSInterfaceData::SITYPE_13;
            STIREQUESTS[2 * i + 1].wRepetitionDelayIn100ms = 0;
135     }

    CViComGSMNWSInterfaceData::SSITypeDemodulationSettings
    cBchDemodulationSettings;

140     cBchDemodulationSettings.wTotalPowerOffsetInDB10 =
        CViComGSMNWSInterfaceData::SSITypeDemodulationSettings
            ::wMinTotalPowerOffsetInDB10;

    cBchDemodulationSettings.wLoadInPercent =
145     CViComGSMNWSInterfaceData::SSITypeDemodulationSettings
        ::wMaxLoadInPercentForTSML;

    cBchDemodulationSettings.sStartMeasurementRequests.dwCountOfSITypeRequests =
    FREQ_COUNT * 2;
150     cBchDemodulationSettings.sStartMeasurementRequests.pSITypeRequest =
    STIREQUESTS;
    cBchDemodulationSettings.sStartMeasurementRequests.dwRequestIdentifier = 0;

    EXEC_VICOM( "Configuring BCH demodulator",
155     pcNws->SetSITypeDemodulationSettings( cError, cBchDemodulationSettings )
    );

    EXEC_VICOM( "Start measurement",
160     pcNws->GetBasicInterface().StartMeasurement( cError ) );

    //
    // The decoding is done until the user manually quits the decoding
    // process by pressing some key.
    //
165     cout << endl << "Measurement is active, press any key to stop it";

    CViComGSMNWSInterfaceData::SMeasResult* pcResult = NULL;
    while ( ( pcResult = pcNws->GetResult( cError, 5000 ) ) )
170     {
        // Check for keyboard stroke to exit demo

```

```

        if ( !_kbhit() )
        {
            break;
        }
175
        if ( ! pcResult->pBCCHDemodulationResult )
        {
            cout << ".";
            continue;
180
        }

        if ( pcResult->pBCCHDemodulationResult->dwRequestIdentifier == 13 )
        {
            cout << "Found GPRS message on channel "
185
                << pcResult->pBCCHDemodulationResult->wFrequencyIndex;
            continue;
        }

        cout << endl << "Channel " << pcResult->pBCCHDemodulationResult-
190
        >wFrequencyIndex;
        cout << " BCCH Type " << pcResult->pBCCHDemodulationResult->eBcchType;
        cout << " Demod Status " << pcResult->pBCCHDemodulationResult-
        >eDemodStatus << endl;

195
        //
        // Every time we receive the result of a system parameters
        // demodulation, we clear the internal cache and start another attempt
        // to decode the message. This is only done to show the usage of the
        // IssueDemodRequest() method, it does not make sense to do so in
200
        // a real environment.
        //
        CViComGSMNWSInterfaceData::S_SIType_Requests::S_SIType_Request
        RESTART_DEMODULATION_REQUEST[] =
        {
205
            // Clear the internal cache
            { pcResult->pBCCHDemodulationResult->wFrequencyIndex,
              CViComGSMNWSInterfaceData::SITYPE_13,
              CViComGSMNWSInterfaceData::SITYPE_DEMOD_BTS,
                0,
210
                pcResult->pBCCHDemodulationResult->dwIndicatorOfSCHInfo },
        };

        CViComGSMNWSInterfaceData::S_SIType_Requests cRequests;
        cRequests.dwCountOfSITypeRequests =
215
            sizeof RESTART_DEMODULATION_REQUEST /
            sizeof CViComGSMNWSInterfaceData::S_SIType_Requests::S_SIType_Request;
        cRequests.pSITypeRequest = RESTART_DEMODULATION_REQUEST;
        cRequests.dwRequestIdentifier = 13;

220
        EXEC_VICOM( "Issuing a special decoding command",
                    pcNws->IssueSITypeRequests( cError, cRequests ) );
    }

    EXEC_VICOM( "Stop measurement",
225
                pcNws->GetBasicInterface().StopMeasurement( cError ) );

    EXEC_VICOM( "Stop measurement",
                pcNws->GetBasicInterface().HasMeasurementStopped( cError, 1000 ) );

230
    EXEC_VICOM( "Unloading Network Scanner...",
                cBchDemodulator.ReleaseTsmx( cError, false ) );
}

/*****/
235
int _tmain(int argc, _TCHAR* argv[])
{
    try
    {
240
        cout << "ViCom GSM BCH Demodulation Demo" << endl << endl << endl;
    }
}

```

```
        RunBchDemodulationSample();
    }
245 catch ( CViComError cError )
    {
        cout << cError.GetErrorString() << endl;
        delete [] cError.GetErrorString();
        return cError.GetErrorCode();
250 }
    return 0;
}
```

The structure of the code is very similar to the samples in the other chapters. Interesting things are done in the following parts of the sample:

- Lines 80-113: The frequency selection is done here. For each frequency, the automatic demodulation is disabled and the minimum measurement configuration is specified (measuring the SCH power). Without the synchronization channel measurement, no demodulation is possible. The default is that the SCH measurement is performed when nothing is specified, so these lines could be omitted here but are included to make this fact clear.
- Lines 120-147: For each channel that has been specified to be measured, two demodulation commands are configured: The one-time demodulation of the SIT 3 message, and the preparation of a BTS specific demodulation command of SIT 13.
- Lines 206-226: Once the SIT 3 has been demodulated, the request to start the demodulation process on the same channel of SIT 13 is started immediately. Therefore the SCH Info indicator is given as id of the BTS that shall be observed.

7 ViCom CDMA 2000 Scanner

The ViCom CDMA 2000 Scanner supports measuring different parameters of a CDMA 2000 network. Such parameters are listed below.

- For the F-PICH:
 - The Channel Impulse Response
 - Absolute Inband Power of the Channel and the Pilot to calculate E_c/I_0 values
 - Automatic peak detection for the CIR measurements
- For the F-SYNC:
 - Configurable demodulation of information sent on the F-SYNC
- General measurement results:
 - Time Estimation of PN arrival time delays
 - Basestation identification

The chapter is organized in three major sections. First, the different measurement modes and how they are configured are described. The sample application is discussed in a separate section, where a descriptive walk-through example is given. Last but not least the chapter closes with a small demo application, showing how to use the API and how to set up a small measurement environment.

7.1 Doing Measurements

Below a description of what kind of measurements are done with the TSMx device is given, and what consequences arise from the implementation are described. In comparison to other ViCom APIs, the different measurement modes are not enabled or disabled directly. The results are provided once they are available.

7.1.1 Measurements

7.1.1.1 Basestation Identification

For each measurement result, an attempt is started to identify the transmitting unit as uniquely as possible. Since it is not possible in all cases to do this automatically in an unambiguous way, the TSMx provides some sort of best effort estimation (see below for a description).

In the result, a basestation is identified by the channel index (i.e. the frequency measured) and a so-called indicator. The indicator is simply a counter that is incremented when a (possibly) new basestation is found on a channel.

If this indicator in combination with the channel index is equal between two measurements, the same BTS has been detected with a high certainty. Under rare circumstances, it might be the case that two different basestations are assigned the same indicator. If the channel is the same, but the indicator is different, it could be the same station, but it can also be another one.

The PN Offset returned besides the channel index and the indicator must not necessarily be the same for two results that belong to the same BTS. It might be the case that in a first step a wrong PN offset was derived from the time estimation, and the correct PN offset has been demodulated later.

The table below summarizes the description above using some example values. There are two columns, one for a possible first and one for the second result. The third column contains the conclusion what can be said about the two results.

Table 4 BTS Identification Sample

First Result			Second Result			Result
Channel	Indicator	PNOffset	Channel	Indicator	PNOffset	
0	0	65535	0	0	64	For the first result, no PN Offset estimation or demodulation has been available, so the offset value is set to invalid. The second attempt seems to provide a better result, either from demodulation or time estimation.
1	154	32	2	154	32	Different BTS, since they were measured on different channels
1	14	31	1	18	98	No certain conclusion can be made, but the cells are very likely to be different.
1	98	87	1	98	89	Probably identical BTS, since channel and indicator are equal, second PN Offset is presumably correct.
1	98	87	1	98	87	Identical BTS when PN offset was demodulated, all values are the same.

7.1.1.2 PN Offset Detection

The identification of a PN offset is done as shown in the figure below. At first, the signal is handled as pseudo-noise if all attempts fail to match the identifying sequence in the signal. The only value reported now is the inband power of the channel.

After some signal power can be separated from the noise and other information in a channel, the CIRs of the F-PICH is are measured and reported. The PN offset is still unknown at that point of time if no prior demodulation has been done. In that case, the PN offset is set to an invalid value (65535). If there has been a successful demodulation on that frequency before, the new incoming signal is set into relation to the previous demodulation result. Based on the arrival time, the scanner makes an assumption what PN offset is assigned to the newly found basestation.

In that phase, the scanner reports Inband power, CIR values and a total power for the signal (RSCP). From these values, the E_c/I_0 can be calculated for the basestation reported.

Once the content of the F-SYNC gets demodulated, the PN offset must not be guessed any more but can be confirmed. The demodulation for a signal can be done in several different ways; see the section below on F-SYNC demodulation settings for more details. The demodulation also is not necessarily performed for all new pilots, so it might be the case that this state is never reached. As an additional result, the content of the F-SYNC is returned by the ViCom API.

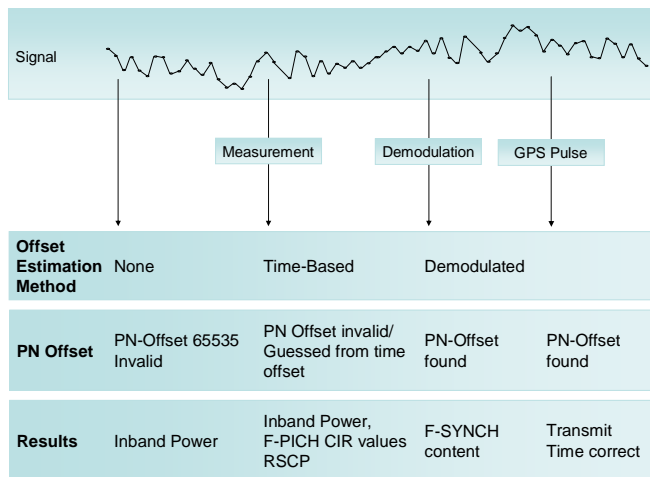


Figure 29 N offset estimation method

7.1.1.3 Channel Impulse Response

One operating mode in the CDMA 2000 scanner is to perform CIR measurements on the F-PICH. In that measurement mode, the TSMx device attempts to demodulate a signal and detect the F-PICH content sent by a basestation on the CDMA channel. When the demodulation and correlation is successful, the TSMx measures at a constant sampling rate power values from the F-PICH and performs a peak analysis on the result.

Such a measurement result is depicted in the figure shown below. For one channel and basestation, the first thing to know is that the samples are all measured at constant intervals. This sampling rate is chosen to be very small, about 0.152 ns.

Since the time plays an important role in a CDMA2000 network, the exact arrival time of the first signal is also contained in the result. The time is specified as offset to the TSMx measurement result time in the number of samples. For example, in the picture below, the delay in number of CIR samples would be 2 (since the second sample after the arrival time is the first valid one).

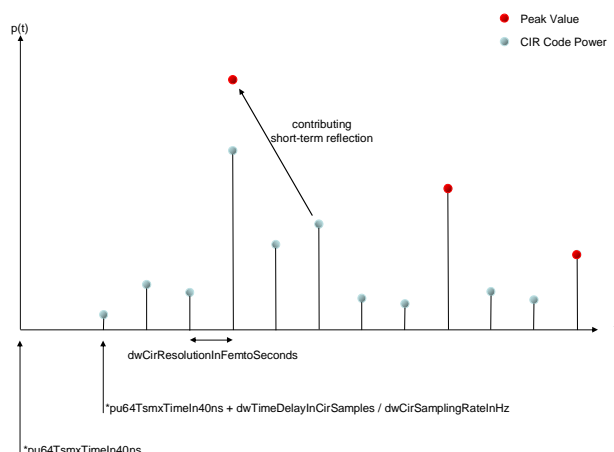


Figure 30 CIR samples from and Peaks

After that point in time, the code power values are measured every `dwCirResolutionInFemtoSeconds` femto seconds. The data recorded in that way can be used to search for peaks that signalize impulse data sent on the F-PICH. This raw data is drawn in light blue circles.

The TSMx software also performs peak detection and automatically includes minor reflection peaks into one that is identified as real impulse that has been sent. In the figure shown above, the first peak found also includes a minor reflection (two steps after the peak occurred). The calculated peak is therefore higher than the original one (it is drawn as red circle, like the two peaks found later).

As another part of the result for the code power measurements the RSCP and inband power values are measured for a detected F-PICH. From both values it is also possible to derive the E_p/I_0 of the pilot. You can do so by subtracting the RSCP from the inband power.



If no CIR measurement can be performed, the TSMx reports the average inband power that has been measured on the channel. It can be found in the `psAverageInbandPowerInDBm100` parameter if that parameter is not NULL.

7.1.1.4 F-SYNC Demodulation

Once the TSMx has detected a pilot on the F-PICH, it tries to synchronize on the F-SYNC and to demodulate the information on that channel. If this is performed successfully, the result is returned in the `sSyncChannelDemodulationResult` structure.

The number of demodulation attempts can be controlled using the `etSyncChannelDemodulationMode` input parameter. For this value, three possible values can be set, each specifying a different kind of demodulation strategy resulting in a different amount of result data.

- `SYNC_CHANNEL_DEMOD_ONCE (1)`: For each channel (frequency), the F-SYNC is only demodulated once. The other PN offsets are therefore calculated by putting the arrival time into relation to the first identified pilot. This can be done since all cells in a CDMA 2000 network are time synchronized and send their information with different offsets.

- `SYNC_CHANNEL_DEMOD_ALL (3)`: When the TSMx detects a new pilot (or assumes to have found a new one), a new demodulation attempt of the F-SYNC is started again. This is a time-consuming task, since the scanner must spend some time to perform the demodulation on a specific pilot signal.
- `SYNC_CHANNEL_DEMOD_FAST (4)`: If a new pilot is found, the demodulation is not performed completely, but in a high-speed fashion. This yields very similar results to the type `SYNC_CHANNEL_DEMOD_ALL`, but does not consume so much time since only a smaller part of the signal must be analyzed.

More details on the content of the demodulation result can be found in the 3GPP2 specification TIA-200.5-D, C.S0005-D. When this document was written, a copy could be found at http://www.3gpp2.org/Public_html/specs/C.S0005-D_v2.0_051006.pdf. See section 3.7.2.3.2.26.

The demodulation result stored in the `pSyncChannelDemodulationResult` parameter contains the following information, besides others:

- Network identification
- The system time and other time information
- The PN offset of the pilot measured
- Channel configuration details

7.1.1.5 Time Estimation

Besides the other results, the TSMx delivers a correction for the time line used for the current measurements. This time estimation is used to correct the time base that is used in a CDMA 2000 system and is specified by single value that defines the currently measured difference between signal and pulse.

The time estimation is based on two different sources: One is the pulse signal from an attached GPS device, the other the demodulation results. Depending what kind of source is currently used to derive the delay value, one of the following three states is active:

- No source available: The transmission time is estimated by setting the received pilots into relation and trying to find a best-match. The delay value cannot be specified.
- Only Demodulation: In this case, the delay is estimated by the received pilots, where the transmission time is still estimated from the different offsets measured. The delay is calculated.
- Demodulation and PPS: Both offsets and transmission time can now be determined, so both values are correct.

7.1.2 Configuration

Before measurements can be done with the CDMA 2000 Scanner, the device has to be configured to prepare the scan process for the desired task. Most of the configuration issues discussed below take influence on all measurement modes, until otherwise noted.

7.1.2.1 Channels and PN Offsets

Before any other type of measurement can be started, it must be defined what channel and what PN Offsets in that channel shall be scanned. A CDMA 2000 Channel is defined by its center frequency in the ViCom API. The set of PN offsets which shall be measured can be defined by enabled or disabling one of 512 boolean attributes in the configuration structure.



The PN offsets that are disabled are not used in the time estimation algorithm as potential candidates for a valid PN. These settings do not influence the demodulation process, so if a PN offset is not set to true in the array, it might nevertheless be demodulated and reported later.

7.1.2.2 Velocity

The TSMx uses a fading correction algorithm that tries to minimize the effect of fading in a moving vehicle. In order to make the algorithm do its job in the best way it can, it needs the maximum speed at which the device will operate. The better this value will match the reality, the better the algorithm will minimize the fading effects.

The maximum speed at which the TSMx is able to operate is 300 km/h (or about 186.4 mph). Using the `sMaxVelocity` structure, the estimated maximum velocity can be set.

7.1.2.3 Calibration

The ViCom API supports the initial calibration of the PPS Pulse Delay with TSMx to improve detection speed and accuracy. Therefore, the value `lMeasuredPPSDelayInNs` parameter in the result structure can be used to specify the measured delay in the `dDelayOfPPSFallingEdgeInSec` configuration parameter. Such a calibration is recommended if you want to measure signals from different networks.

To perform the calibration, it is important to know the PN offset and the distance to a specific basestation. Such a basestation signal is measured with the TSMx device, and based on the timing found during that initial measurement the PPS delay can be calculated.

From the distance d_{PN} [m] and the PN offset $offset_{t_{PN}}$, the expected time delay T_{exp} can be calculated as follows:

$$T_{exp} = offset_{t_{PN}} * 64 / 1228800 \text{ sec} + d_{PN} / (300 \text{ m/sec})$$

This expected time offset can then be compared with the measured time offset T_{meas} . The difference of both values is the time delay that must be subtracted from measurements and the GPS pulse signal. To let the TSMx automatically adjust the GPS pulse, you have to set the time delay in the `iDelayOfPPSFallingEdgeInSec` configuration setting.

7.2 Sample Application

As in the previous chapters, the CDMA 2000 functionality and handling of the ViCom interface can be tested in the distributed sample application. The goal of the user interface design of the sample application is not to have maximum comfort or to make it available to end-users, but to provide all possible configurations and operations to a developer who wants to find out how the API is used.

Below you can see a screenshot taken from the application. The left side of the dialog shows a set of buttons that can be used to perform the various operations on the TSMx. Many of them transfer a specific configuration setting that has been modified in the dialog to the device, others control the current state (i.e. measurement mode active or not).

On the right side, some common settings and result viewing controls can be found. Settings can be stored on the harddisk as well, using the Load and Save mechanisms in that part of the dialog.

The biggest part of the dialog is occupied by controls that specify measurement settings of the CDMA 2000 scanner. Besides the left-most list of frequencies that shall be scanned, a control that can be used to enable and disable the scanning of a specific PN Offset is the most obvious element.

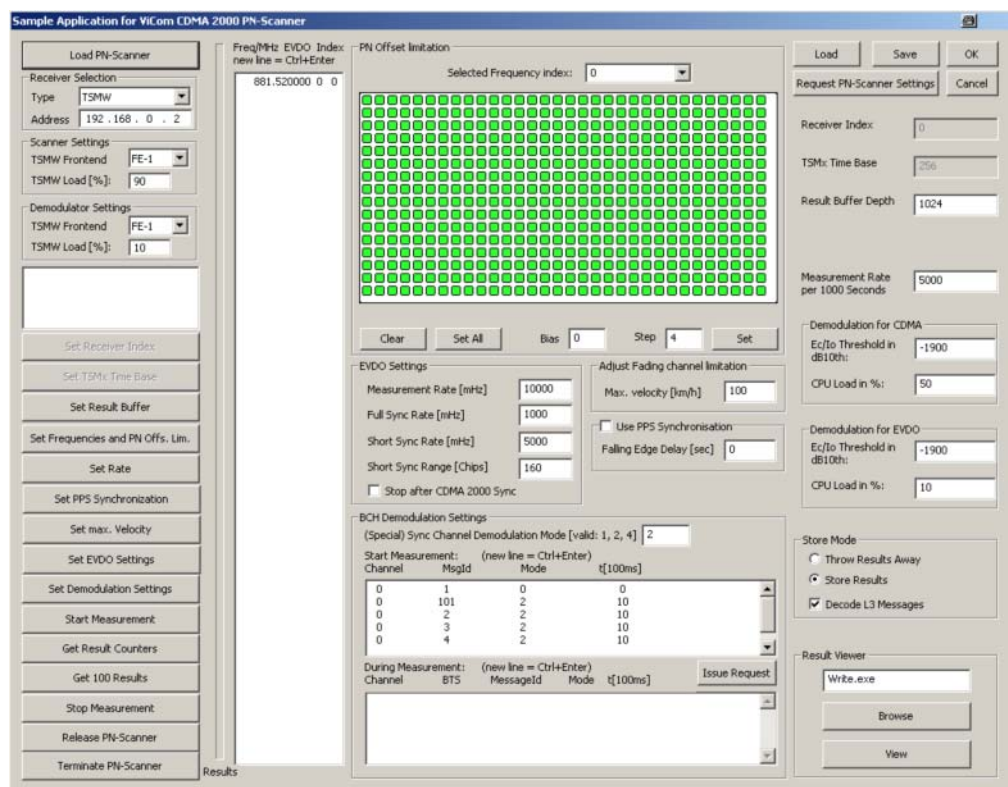


Figure 31 CDMA 2000 Scanner Sample Application

Below that element, some CDMA 2000 specific settings can also be specified. These measurement settings are described in the following sections.

7.2.1.1 Setting the PN Offsets for time estimation

To specify which PN Offsets shall be used for time estimation during scans, a big switch matrix is used in the sample application. Each rectangle in the matrix that is shown in the upper center of the dialog stands for one PN Offset. Green signalizes an enabled scan, where a light gray filling stands for disabled reporting.

The upper left box is used to control estimation of PN offset 0, the other offsets are first set from left to right (offsets 0-31 in the first line), then from top to bottom.

You can simply enable/disable single points by clicking on them. To modify all entries at once, use the two buttons below the matrix on the left set. On the right side, there is a function to enable a certain pattern of PN offsets. The value stored in "Bias" is the starting offset, and "Step" is the distance between two different offsets that shall be enabled. For example, if you choose "Bias" to be 13 and "Step" to be 7, PN offsets 13, 20, 27, 34, ... and so on will be enabled after you click on "Set".



Note that the whole content will be erased before the "Bias"/"Step" settings are applied using the "Set" button. Manipulations that were made before are cleared.

7.2.1.2 Sync Channel Demodulation Mode

In this field, you can specify how often an attempt is started to demodulate a probably new incoming signal. The values that can be put in there can either be 1, 2 or 4. The meaning of the numeric value is the same as described above in the F-SYNC Demodulation section. The value 4 allows fast and highly sensitive detection of pilots and might therefore be a good default setting.

7.2.1.3 Performing Measurements

To perform measurements with the demo application, some steps must be performed in a specific sequence. If you follow the list of actions described below, this should help you to circumvent pitfalls, when programming your own applications with the API.

1. Load the TSMx device by clicking on "Load PN-Scanner" button. Assure that the driver is loaded successfully and information from the loaded device is shown in the box below.
2. Click the buttons "Set Receiver Index", "Set TSMx Time Base", "Set Result Buffer" and "Set Rate" to apply the related settings shown on the right-hand side of the dialog. Change them before clicking the buttons if you want different values than the defaults to be applied.

3. Enter the frequencies that shall be scanned in the appropriate list. You can add multiple frequencies by pressing <Ctrl>+<Enter> to add a new line to the list. The channel indices are optional, you can omit them to let the sample application create them automatically.
In the same step you have to define which PN offsets shall be detected using the time estimation result. This does not influence demodulation, so if you choose that all pilots shall be demodulated (setting 4 in the related field), then each PN offset should at some point be the result of a successful demodulation, even if the field was disabled in the matrix.
Click "Set Frequencies" to finish that step.
4. Apply the CDMA 2000 scanner specific settings using the buttons "Set PPS Synchronisation", "Set max.Velocity" and "Set Sync Channel Demod. Mode". Do your modifications to these settings before, for example, perform a measurement calibration to put a meaningful value into the "Delay of PPS falling edge" field.
5. Start the measurement using "Start Measurement" button.
6. Choose whether you want to store the results in a text file and what kind of information shall be saved. Then you can click on "Get Results" to write 100 measurement results into the file. When this is done, the application is blocked until 100 results have been measured (unless older results are still in the buffer).
7. Once you have all your data, click "Stop Measurement" button to stop measurement again. You can then either change the settings (steps 2, 3 and 4) or you can unload the TSMx device ("Release PN-Scanner") and quit the application.

7.3 Sample Code

Finally some sample code is shown in this section that starts a CDMA 2000 session and performs some basic measurement tasks. The main goal in the sample application is to show the usage of the CDMA 2000 ViCom API. In the sample, the CIR measurement peak results are used to generate a list of currently detected basestations, and give feedback every five seconds what stations have been found and which are the strongest.

After the code, a short discussion is given on what the different sections in the code actually do. The sample code is structured in the same way as the other samples in this manual, so you should be familiar with the style when you read one of the other chapters.

```

#include "stdafx.h"

#include <windows.h>
#include <iostream>
5  #include <conio.h>

#include <algorithm>
#include <vector>

10 #include "ViComBasicInterface.h"
#include "ViComError.h"
#include "ViComLoader.h"
#include "ViComCdma2000Errors.h"
#include "ViComCdma2000Interface.h"
15 #include "ViComCdma2000InterfaceData.h"

```

```

using namespace std;

20  /*****
//
// This macro is just a convenience function to reduce the amount of code
// in the sample and to focus on the more important aspects. Error handling
// is done here as well.
25  //
#define EXEC_VICOM( msg, call ) \
    cout << msg << "..."; \
    call; \
    if ( cError.GetErrorCode() > 0 ) { \
30  char* szErr = new char[strlen(cError.GetErrorString()) + 1]; \
    strcpy(szErr, cError.GetErrorString()); \
    throw CViComError(cError.GetErrorCode(), szErr); } \
    cout << "ok" << endl;

35  /*****
//
// We need a structure to keep track of the currently detected
// basestations and the highest CIR peak measured.
40  //
struct sMaxCir
{
    WORD nPnOffset;
    WORD nIndicator;
45  DWORD nChannel;
    short nCirMax;
};

50  /*****
//
// This function compares two sMaxCir objects and returns whether the
// first object is greater than the second (according to the
// max peak value stored in both objects), because we want the
55  // strongest sender first.
//
bool CompareMaxCirs( sMaxCir& sMaxCir1, sMaxCir& sMaxCir2 )
{
60  }
    return sMaxCir2.nCirMax < sMaxCir1.nCirMax;
}

/*****
void RunCdma2000Sample()
65  {
    CViComError cError;
    CViComLoader<CViComCdma2000Interface> cVicomApiLoader;

70  //
// Below is the standard initialization sequence of a ViCom module. We first
// try to load the DLL, then the device. If both is successful, we can query
// the loading mechanism class for a pointer onto the controlling
// CDMA 2000 API.
//
75  EXEC_VICOM( "Loading CDMA 2000 Scanner",
    cVicomApiLoader.LoadTsmx( cError ) );

    CViComCdma2000Interface* pCdma2000If;
    EXEC_VICOM( "Retrieving special interface",
80  pCdma2000If = cVicomApiLoader.GetpInterface( cError ) );

//
// Common initialization sequence. In this section, the very basic settings
// of the TSMx device are specified. These are (in the order of the related
85  // method calls): The TSMx device in the receiver chain (0 is the first
// device connected via Firewire), the time synchronisation mode and the

```

```

// number of results that are stored in the internal buffer. The last one
// is set to the maximum possible value, but you can change this as desired.
//
90 EXEC_VICOM( "Setting receiver index",
    pCdma2000If->GetBasicInterface().SelectReceiver( cError, 0 ) );

//
// If a more precise time synchronisation is desired, connect a GPS device
95 // to the TSMx and choose TIME_BASE_SYNC_MODE_PPS
//
EXEC_VICOM( "Setting time base",
    pCdma2000If->GetBasicInterface().SetTimebaseSynchronisationMode( cError,
    CViComBasicInterfaceData::TIME_BASE_SYNC_MODE_INTERNAL ) );
100
    CViComBasicInterfaceData::SResultBufferDepth cResultBufferDepth;
    cResultBufferDepth.dwValue =
CViComBasicInterfaceData::SResultBufferDepth::dwMax;

105 EXEC_VICOM( "Setting result buffer",
    pCdma2000If->GetBasicInterface().SetResultBufferDepth( cError,
    cResultBufferDepth ) );

//
// After the device has been loaded, it must be configured for the specific
// measurement task(s). Common for all tasks is the definition of what
110 channels
// and frequencies must be measured, and how the measurement is actually
// performed (depends also in the device).
//
115 //
//
// We try to scan on three channels, and want every possible PN offset
// to be reported. Therefore we have to enable all entries in the
120 // bTableOfPNOffsetArbitraryLimitation table by settings them to TRUE.
//
static const double CHANNEL_FREQUENCIES[] = { 2110.05 };
const int CHANNEL_COUNT = sizeof CHANNEL_FREQUENCIES / sizeof
CHANNEL_FREQUENCIES[0];
125
    CViComCdma2000InterfaceData::SFrequencySetting vcFreqSettings[ CHANNEL_COUNT
];
    for ( int i=0; i<CHANNEL_COUNT; i++ )
    {
130         vcFreqSettings[i].dCenterFrequencyInMHz = CHANNEL_FREQUENCIES[i];
        vcFreqSettings[i].bIsEvdoFrequency = FALSE;
        for ( int j=0;
j<CViComCdma2000InterfaceData::SFrequencySetting::dwFixCountOfPNOffsetArbitraryL
imits; j++ )
135         {
            vcFreqSettings[i].bTableOfPNOffsetArbitraryLimitation[j] = TRUE;
        }
    }

140 CViComCdma2000InterfaceData::SChannelSettings cChannelSettings;
cChannelSettings.dwCount = CHANNEL_COUNT;
cChannelSettings.pdTableOfFrequencySetting = vcFreqSettings;

EXEC_VICOM( "Setting frequency table",
145         pCdma2000If->SetFrequencyTable( cError, cChannelSettings ) );

//
// Now we specify the measurement mode. We use MEAS_MODE_HIGH_SPEED here to
// make this demo run on every ViCom supporting TSMx device. The measurement
150 // rate is also defined to be the highest supported on all devices.
//
CViComCdma2000InterfaceData::SMeasurementRate cMeasRate;
cMeasRate.dwValuePer1000Sec =
CViComCdma2000InterfaceData::SMeasurementRate::dwMaxPer1000SecForTSMx;
155 EXEC_VICOM( "Setting measurement mode",
    pCdma2000If->SetMeasurementRate( cError, cMeasRate ) );

```

```

160     //
    // For the sample, we don't use any PPS signals to make it work in the
    // very basic environment. If you have a GPS receiver attached to the
    // selected TSMx device, you can change this to the delay of the
    // GPS signal found with a calibration measurement (see manual).
    //
165     CViComCdma2000InterfaceData::SPPSSettings cPulseSettings;
    cPulseSettings.iDelayOfPPSFallingEdgeIn100ns =
CViComCdma2000InterfaceData::SPPSSettings::iInvalidPPSDelayIn100ns;

    EXEC_VICOM( "Setting PPS configuration",
170         pCdma2000If->SetPPSSettings( cError, cPulseSettings ) );

    //
    // The internal fading correction algorithm can be configured by
    // specifying the maximum speed of the test vehicle. It is specified
175     // in kilometers per hour (see reference for a conversion to mph).
    //
    // The better the estimation of the final speed is, the better the
    // fading algorithm will work.
    //
180     CViComCdma2000InterfaceData::SMaxVelocity cMaxVelocity;
    cMaxVelocity.dMaxVelocityInKmPerHour =

CViComCdma2000InterfaceData::SMaxVelocity::dwMaxVelocityInKmPerHourDefault;

185     EXEC_VICOM( "Setting the maximum velocity",
        pCdma2000If->SetMaxVelocity( cError, cMaxVelocity ) );

    //
    // We want to scanner to demodulate all new pilots it finds, since
190     // it is our desire to get the maximum of information available in an
    // unknown network environment.
    //
    EXEC_VICOM( "Setting F-SYNC demodulation type",
195         pCdma2000If->SetSyncChannelDemodulationMode(
            cError,
            CViComCdma2000InterfaceData::SYNC_CHANNEL_DEMOD_ALL ) );

    //
    // All configuration is done, so the measurement can be started.
200     //
    EXEC_VICOM( "Start measurement",
        pCdma2000If->GetBasicInterface().StartMeasurement( cError ) );

    //
205     // In this demo application, we do a simple air-scanning for
    // CDMA 2000 signals and if we find any BTS, we write some footprint
    // data onto the stdout, until the user stops the scanning by
    // pressing any key.
    //
210     cout << endl << "Measurement is active, press any key to stop it";

    std::vector< sMaxCir > vsMaxCirs;
    DWORD dwLastOutput = GetTickCount();

215     CViComCdma2000InterfaceData::SMeasResult* pcResult = NULL;
    while ( ( pcResult = pCdma2000If->GetResult( cError, 1000 ) ) )
    {
        // Check for keyboard stroke to exit demo
220         if ( _kbhit() )
        {
            break;
        }

        //
225         // We report the inband power currently measured to show
        // some kind of constant progress in this sample.
        //
        if ( pcResult->psAverageInbandPowerInDBm100 )

```



```

230     {
        cout
            << "Channel: "
            << pcResult->dwChannelIndex
            << " Inband power: "
            << 0.01* (*pcResult->psAverageInbandPowerInDBm100)
235         << endl;
    }

    //
    // Report if a new pilot demodulation has been perform
    // successfully.
    //
240     if ( pcResult->pSyncChannelDemodulationResult )
    {
245         cout << "Demodulation succeeded for PN Offset: ";
        cout << pcResult->pSyncChannelDemodulationResult->wPILOT_PN;
        cout << endl;
    }

    //
250     // If we find a CIR measurement in the result, we scan the list of peaks
    // for the maximum value and compare it to the
    //
    SViComList<CViComCdma2000InterfaceData::SMeasResult::SFPiChCir>::SLinkedObject*
255     pFPiChCirResult = pcResult->ListOfFPiChCirs.pFirst;
    while ( pFPiChCirResult )
    {
        //
        // First we search for the maximum CIR within the peaks of the
260     currently
        // reported basestation.
        //
        SViComList<CViComCdma2000InterfaceData::SMeasResult::SFPiChCir::SPeakInfo>::SLink
265     kedObject*
        pFPiChPeak = pFPiChCirResult->ListOfPeaks.pFirst;
        short nMaxCirPeak =
        CViComCdma2000InterfaceData::SMeasResult::sInvalidPowerInDBm100;
270         while ( pFPiChPeak )
        {
            nMaxCirPeak = max( nMaxCirPeak, pFPiChPeak->sPeakPowerInDBm100 );
            pFPiChPeak = pFPiChPeak->pNext;
        }

275         //
        // Now we search for an entry in our structure that keeps track of
        // the highest CIR peak found for each basestation. If there is one
        found,
        // it is updated with the new peak if the new peak is higher than the
280         // one currently stored.
        //
        for ( unsigned int i=0; i<vsMaxCirs.size(); i++ )
        {
285             if ( vsMaxCirs[i].nPnOffset == pFPiChCirResult-
                >ExtendedPNOffset.wPNOffset
                && vsMaxCirs[i].nIndicator == pFPiChCirResult-
                >ExtendedPNOffset.wIndicator
                && vsMaxCirs[i].nChannel == pcResult->dwChannelIndex )
            {
290                 vsMaxCirs[i].nCirMax = max( vsMaxCirs[i].nCirMax, nMaxCirPeak );
                break;
            }
        }

295         //
        // If all entries were checked but no entry matched the currently
        // reported basestation, we create a new entry and initialize it.
        //
        if ( i == vsMaxCirs.size() )

```



```

300         {
            sMaxCir sNewMaxCir;
            sNewMaxCir.nPnOffset = pFPiChCirResult->ExtendedPNOffset.wPNOffset;
            sNewMaxCir.nIndicator = pFPiChCirResult-
>ExtendedPNOffset.wIndicator;
305         sNewMaxCir.nChannel = pcResult->dwChannelIndex;
            sNewMaxCir.nCirMax = nMaxCirPeak;
            vsMaxCirs.push_back( sNewMaxCir );
        }

310     pFPiChCirResult = pFPiChCirResult->pNext;
    }

    //
    // Print current CIR power ranking every 5 seconds
    //
315     if ( GetTickCount() - dwLastOutput > 5000 )
    {
        //
        // Sort the unordered results in the vsMaxCirs vector by
        // the maximum power peak and print the output
        // in that sequence.
        //
320     sort( vsMaxCirs.begin(), vsMaxCirs.end(), CompareMaxCirs );

325     cout << endl << "CIR Status: " << endl;
        for ( unsigned int i=0; i<vsMaxCirs.size(); i++ )
        {
            cout << "\tPN " << vsMaxCirs[i].nPnOffset;
            cout << " Channel " << vsMaxCirs[i].nChannel;
330     cout << " ID " << vsMaxCirs[i].nIndicator;
            cout << " Max Pow: " << 0.01 * vsMaxCirs[i].nCirMax;
            cout << endl;
        }
        cout << endl;

335     dwLastOutput = GetTickCount();
        vsMaxCirs.clear();
    }
}

340 //
// Measurement has been stopped, so we properly shutdown the
// demo application by stopping the measurement and unloading the
// device. In a more sophisticated software, these two steps might
345 // not necessarily be executed at the same time.
//
EXEC_VICOM( "Stop measurement",
            pCdma2000If->GetBasicInterface().StopMeasurement( cError ) );

350 EXEC_VICOM( "Waiting for measurement to stop",
            pCdma2000If->GetBasicInterface().HasMeasurementStopped( cError, 1000 ) );

EXEC_VICOM( "Unloading PN Scanner...",
355     cVicomApiLoader.ReleaseTsmx( cError, false ) );
}

/*****

360 int _tmain(int argc, _TCHAR* argv[])
    {
        try
        {
            cout << "ViCom CDMA 2000 Scanner Demo" << endl << endl << endl;
365     RunCdma2000Sample();
        }
        catch ( CViComError cError )
        {
370     cout << cError.GetErrorString() << endl;

```

```
        delete [] cError.GetErrorString();  
        return cError.GetErrorCode();  
    }  
375     return 0;  
    }
```

The code is structured in the same way as the other examples in this manual. To avoid confusion and to focus on the usage of the ViCom API, only one central function contains all the application logic. This function starts in line 64 and lasts to line 347. Most of the code in the function is dedicated to the configuration of the scanner. Every available option is specified in this sample, but it is not necessary to do the same in your own application. The most important settings are done in lines 120-142, where the channel configuration is defined. In these lines, the frequency to be scanned is configured. You can add other frequencies simply by extending the frequency array in line 120.

Some CDMA 2000 specific settings are defined in the lines 162-167 and 189-192, where pulse settings and F-SYNC demodulation type are set. The former is also used when the PPS delay is calibrated, in the sample it simply is set to the default value of 0 (which means no delay). The latter option is discussed in detail in section “F-SYNC Demodulation” on page 101.

In lines 212-331 the measurement loop can be found. The scanner is asked constantly for new measurement results, and if some are available, the application analyses them. As described shortly before, the main goal of the demo program is to create a list of best stations every five seconds. Therefore, the peak results of the CIR measurements are used to find the strongest measured power value for a BTS (258-267). The currently found basestation list and their highest detected power value are stored in a simple vector and updated with newly found information in lines 275-286. If the BTS found is not in the list, a new entry is added to the list (lines 292-300). Once the results have been analysed, they are written to standard out if the last result has been printed at least 5 seconds earlier (308-329). This is done until the user explicitly ends the program (by pressing a key) or when no result has been measured for one second (which should not be the case normally).

7.4 CDMA 2000 BCH Demodulation

Like other scanners of the TSMx product family, the CDMA 2000 scanner supports demodulation of specific messages transferred on the scanned channels. The basic concept is described in the chapter “Using the Demodulators”.

In this section, the extended sample application and a sample code is shown how the demodulator is used in the CDMA 2000 scanner environment. In comparison to the WCDMA scanner, there are no major differences to the general concept.

7.4.1 Measurement Algorithm

In comparison to the other demodulation strategies, the CDMA 2000 demodulator has little information where to find the actual messages on the channel. There are little guarantees defined in the specification. To avoid blocking the TSMx device when trying to perform a demodulation for long time, the demodulator uses a randomized algorithm that tries to detect messages in smaller subsets of the (repeated) signal, accumulating the results and trying to distribute the measured subsets measurements as uniformly over the signal as possible.

This is a good working approach to demodulate the CDMA 2000 messages. In a CDMA 2000 system, the messages can be found quite well by checking a sequence due to the improved signal to noise distance (compared to other technologies like GSM which enforce a pre-defined structure of messages).

Besides that, CDMA 2000 provides another speciality in the ViCom demodulator API. The information is extracted from different channels, like the SYNC channel, BCCH and PCH. This fact is used in two ways: On one side, the demodulator can be told to only extract messages on a subset of the channels, on the other side the channel on which the message is found is returned in the demodulation result.



Note that regardless of the demodulation settings, the synch channel message information is always delivered as first demodulation result (even if no special demodulation is configured at all). The content is extracted after synchronization and is therefore available as first result.

7.4.2 Sample Application

The sample application as shown in the previous section has been extended to also support the demodulation request configuration before and during the measurement. Structure and usage are very similar to the demodulator extensions described in "BCH demodulator - Programming Sample" of the "ViCom WCDMA PN Scans" chapter on page 43 and the related GSM equivalent.

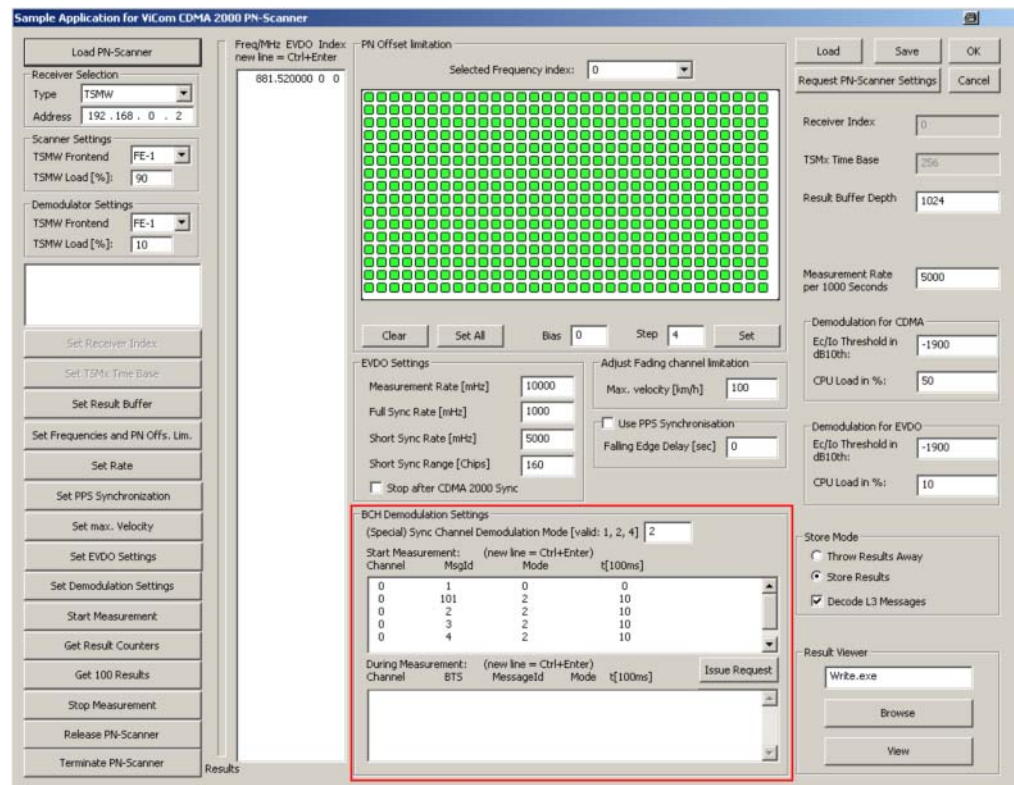


Figure 32 Sample Application CDMA 2000 Demodulator

As depicted in the figure above, the CDMA 2000 sample application GUI has been changed to make it possible to define the demodulation settings in two ways: The upper list in the red box is used to configure initial demodulation settings before a measurement is started, and the lower one to define demodulation commands during measurement.

A detailed description how to use that controls is given in WCDMA Scanner Sample Application on page 55 and GSM GSM BCH Demodulation on page 91. Please read these related sections to find out the demodulator is configured.

7.4.3 Code Example

Down below a code listing shows the usage of the demodulator from the developers' point of view. You can use it as a starting point when starting implementation. The code is documented and comparable to the sample shown in the chapter BCH Demodulation, section Programming Sample.

The sample is kept in the same style already known from other samples in this manual, so the description of the details is focussed on the main parts of the code.

```

#include "stdafx.h"

#include <conio.h>
#include <iostream>
5 #include <windows.h>

#include "ViComBasicInterface.h"
#include "ViComError.h"
#include "ViComLoader.h"

```

```

10  #include "ViComCdma2000Errors.h"
    #include "ViComCdma2000Interface.h"
    #include "ViComCdma2000InterfaceData.h"

15  using namespace std;

    /*****

20  #define EXEC_VICOM( msg, call ) \
        cout << msg << "..."; \
        call; \
        if ( cError.GetErrorCode() > 0 ) { \
            char* szErr = new char[strlen(cError.GetErrorString()) + 1]; \
            strcpy(szErr,cError.GetErrorString()); \
            throw CViComError(cError.GetErrorCode(), szErr); } \
25  cout << "ok" << endl;

    /*****

30  void RunBchDemodulationSample()
    {
        CViComError cError;
        CViComLoader< CViComCdma2000Interface > cBchDemodulator;

35  //
        // Below is the standard initialization sequence of a ViCom module.
        //
        EXEC_VICOM( "Loading CDMA 2000 Scanner",
40  cBchDemodulator.LoadTsmx( cError ) );

        EXEC_VICOM( "Retrieving special interface",
            CViComCdma2000Interface* pcC2kScanner = cBchDemodulator.GettpInterface(
                cError ) );

45  //
        // Common initialization sequence
        //
        EXEC_VICOM( "Setting receiver index",
50  pcC2kScanner->GetBasicInterface().SelectReceiver( cError, 0 ) );

        EXEC_VICOM( "Setting time base",
            pcC2kScanner->GetBasicInterface().SetTimebaseSynchronisationMode( cError,
                CViComBasicInterfaceData::TIME_BASE_SYNC_MODE_PPS_WATCHED_BY_GSM ) );

55  CViComBasicInterfaceData::SResultBufferDepth cResultBufferDepth;
        cResultBufferDepth.dwValue =
        CViComBasicInterfaceData::SResultBufferDepth::dwMax;
        EXEC_VICOM( "Setting result buffer",
60  pcC2kScanner->GetBasicInterface().SetResultBufferDepth( cError,
            cResultBufferDepth ) );

        //
        // The measurement rate and mode must be specified to produce a
        // meaningful measurement result.
65  // Set measurement mode to the smallest common divisor, that is
        // the minimum supported by the TSML.
        //
        CViComCdma2000InterfaceData::SMeasurementRate cMeasRate;
        cMeasRate.dwValuePer1000Sec =
70  CViComCdma2000InterfaceData::SMeasurementRate::dwMaxPer1000SecForTSML;
        EXEC_VICOM( "Setting Measurement Details",
            pcC2kScanner->SetMeasurementRate( cError, cMeasRate ) );

        //
75  // Now a set of frequencies is defined on which a BCH may be found. This
        might
        // vary in different countries and for different network providers, so the
        // value can be changed easily.
        // In this example, this is just a test frequency used in the signal
80  simulator.

```

```

//
// Note that depending on the device attached to the system, the maximum
number
// of frequencies differs. For the TSML-C, it is 6, for example.
85 //
//
static const double CHANNEL_FREQUENCIES[] = { 2110.6 };
const int CHANNEL_COUNT = sizeof CHANNEL_FREQUENCIES / sizeof
CHANNEL_FREQUENCIES[0];

90 CViComCdma2000InterfaceData::SFrequencySetting vcFreqSettings[ CHANNEL_COUNT
];
for ( int i=0; i<CHANNEL_COUNT; i++ )
{
95   vcFreqSettings[i].dCenterFrequencyInMHz = CHANNEL_FREQUENCIES[i];
   vcFreqSettings[i].bIsEvdoFrequency = FALSE;
   for ( int j=0;
j<CViComCdma2000InterfaceData::SFrequencySetting::dwFixCountOfPNOOffsetArbitraryL
imits; j++ )
100   {
       vcFreqSettings[i].bTableOfPNOOffsetArbitraryLimitation[j] = TRUE;
   }
}

CViComCdma2000InterfaceData::SChannelSettings cChannelSettings;
105 cChannelSettings.dwCount = CHANNEL_COUNT;
cChannelSettings.pdTableOfFrequencySetting = vcFreqSettings;

EXEC_VICOM( "Setting frequency table",
110   pcC2kScanner->SetFrequencyTable( cError, cChannelSettings ) );

//
// Now the demodulation details can be defined. Since the current CDMA 2000
// demodulator supports decoding the system parameters message only, we
request
// a demodulation here.
115 //
//
CViComCdma2000InterfaceData::S_DemodRequests::S_DemodRequest DEMOD_REQUESTS[ ]
=
120 {
    // Decode the system parameters message for each bts only once
    { 0,
      { CViComCdma2000InterfaceData::C2K_MSGID_SYS_PARAMS,
        (CViComCdma2000InterfaceData::etChannelType)
(CViComCdma2000InterfaceData::C2K_CH_TYPE_BCCH |
125 CViComCdma2000InterfaceData::C2K_CH_TYPE_PCH) },
        CViComCdma2000InterfaceData::DEMOD_ONCE, 0, 0 },
    };

    CViComCdma2000InterfaceData::SBchDemodulationSettings
130 cBchDemodulationSettings;
    cBchDemodulationSettings.wLoadInPercent =

    CViComCdma2000InterfaceData::SBchDemodulationSettings::wMaxLoadInPercentForTSML;
    cBchDemodulationSettings.lEcToIoThresholdInDB100 =
135 CViComCdma2000InterfaceData::SBchDemodulationSettings::lminEcToIoThresholdInDB10
0ForTSML; // == -10 dB

    cBchDemodulationSettings.sStartMeasurementRequests.dwCountOfRequests =
    sizeof DEMOD_REQUESTS / sizeof
140 CViComCdma2000InterfaceData::S_DemodRequests::S_DemodRequest;
    cBchDemodulationSettings.sStartMeasurementRequests.pRequests =
    DEMOD_REQUESTS;

    EXEC_VICOM( "Configuring BCH demodulator",
145   pcC2kScanner->SetBchDemodulationSettings( cError, cBchDemodulationSettings
) );

    EXEC_VICOM( "Start measurement",
150   pcC2kScanner->GetBasicInterface().StartMeasurement( cError ) );

//

```

```

// The decoding is done until the user manually quits the decoding
// process by pressing some key.
//
155 cout << endl << "Measurement is active, press any key to stop it";

CViComCdma2000InterfaceData::SMeasResult* pcResult = NULL;
while ( ( pcResult = pcC2kScanner->GetResult( cError, 5000 ) ) )
{
160 // Check for keyboard stroke to exit demo
    if ( _kbhit() )
    {
        break;
    }

165 // if ( ! pcResult->pMsgResult )
    {
        cout << ".";
        continue;
    }

170 // cout << endl << "Channel " << pcResult->dwChannelIndex;
    cout << ", BTS " << pcResult->pMsgResult->wBtsId;
    cout << " / First ID: " << pcResult->pMsgResult->wFirstBtsId ;
175 cout << ", Channel Type: " << pcResult->pMsgResult->MeasSpec.ChannelType
    << endl;

    //
    // Every time we receive the result of a system parameters
    // demodulation, we clear the internal cache and start another attempt
    // to decode the message. This is only done to show the usage of the
    // IssueDemodRequest() method, it does not make sense to do so in
    // a real environment.
    //
185 CViComCdma2000InterfaceData::S_DemodRequests::S_DemodRequest
    RESTART_DEMODULATION_REQUEST[] =
    {
        // Clear the internal cache
        { 0,
190 { CViComCdma2000InterfaceData::C2K_MSGID_SYS_PARAMS,
CViComCdma2000InterfaceData::C2K_CH_TYPE_ALL },
CViComCdma2000InterfaceData::DEMOM_NODE_B_RESET, 0, pcResult-
>pMsgResult->wBtsId },
        // And request the demodulation again
195 { 0,
{ CViComCdma2000InterfaceData::C2K_MSGID_SYS_PARAMS,
CViComCdma2000InterfaceData::C2K_CH_TYPE_ALL },
CViComCdma2000InterfaceData::DEMOM_NODE_B, 0, pcResult->pMsgResult-
>wBtsId },
200 };

    CViComCdma2000InterfaceData::S_DemodRequests cRequests;
    cRequests.dwCountOfRequests = 0;
    sizeof RESTART_DEMODULATION_REQUEST /
205 sizeof CViComCdma2000InterfaceData::S_DemodRequests::S_DemodRequest;
    cRequests.pRequests = RESTART_DEMODULATION_REQUEST;

    EXEC_VICOM( "Issuing a special decoding command",
210 pcC2kScanner->IssueDemodRequests( cError, cRequests ) );
}

EXEC_VICOM( "Stop measurement",
pcC2kScanner->GetBasicInterface().StopMeasurement( cError ) );

215 EXEC_VICOM( "Waiting for measurement to stop",
pcC2kScanner->GetBasicInterface().HasMeasurementStopped( cError, 1000 ) );

EXEC_VICOM( "Unloading Network Scanner...",
220 cBchDemodulator.ReleaseTsmx( cError, false ) );
}

/*****

```

```
225 int _tmain(int argc, _TCHAR* argv[])
    {
        try
        {
            cout << "ViCom CDMA 2000 Demodulation Demo" << endl << endl << endl;
230     RunBchDemodulationSample();
        }
        catch ( CViComError cError )
        {
235     cout << cError.GetErrorString() << endl;
            delete [] cError.GetErrorString();
            return cError.GetErrorCode();
        }
240     return 0;
    }
```

The interesting sections of the code are described in here:

- Lines 82-104: Here the basic setup of the measurement is performed. In this example, the (virtual) CDMA 2000 channel at 2110.6 MHz shall be scanned for a signal. The demodulator shall attempt to extract the system parameters message. The PN Offset of the channel is not handled in the example, that's why the scanner is not restricted in that area.
- Lines 111-135: The demodulation settings are set in these lines. Here, the type of the message that shall be demodulated is set, as is the kind of channel shall be used to find the message. In this case, there is no restriction.
- Lines 174-198: In this example, the repetition demodulation mode is imitated by resetting the demodulation result once it has been retrieved, and a new request of the same kind is started. Of course, this is nothing you would do in a real world application, but it is used here to demonstrate the usage of the different API commands.

7.5 EVDO Measurements

Within the CDMA 2000 Scanner API the EVDO measurement capabilities are embedded. Only few additional control settings are required to do a proper setup of an EVDO measurement. The basic initialization has to be done as shown in the CDMA 2000 example above. However, there are two major differences:

First, the frequencies that shall be interpreted as EVDO measurement have to be marked as such. The `blsEvdoFrequency` attribute has to be set to `TRUE` therefore. Due to the fact that the measurement is done in a different way compared to the CDMA 2000 scan, the settings that control the measurement algorithms are a little different. A special settings structure is available to control all EVDO related parameters. Therefore the second difference is to specify the content of the `SEvdoControlSettings` structure.

7.5.1 The SEvdoControlSettings structure

The EVDO measurements can only be performed in a proper way when a CDMA 2000 channel is measured as well. In this way, the time relationship between the different PN offsets is resolved. If no additional CDMA 2000 measurements have to be performed, the CDMA 2000 measurement can be disabled after the first successful demodulation. This is done by setting the `bStopCdma2000AfterSync` attribute to true.

Time offset estimation is done in a quite different way compared to the CDMA 2000 measurements, and three of the five attributes of the structure are dealing with that subject. Once a signal of a basestation has been detected, the scanner knows in principle when the arrival of the next information can be expected, and only processes that subrange of the incoming data to find new information. In regular intervals, the subrange has to be extended to either find signals from new basestations or to compensate shifts due to fading or movement of the measurement device.

In these cases, the processing overhead is increased. To save processing time, two different types of subrange extensions are done: A short sync and a full sync. The short sync also only uses a subset of chips that are part of a PN offset timeslot. The size of the subrange is specified by the `dwShortSyncRangeInChips` parameter, and the frequency of the short sync `dwShortSyncRateInmHz` in milliHertz. The full sync is (normally) done less frequently, and uses all the available data for signal recognition. The frequency of that task is specified in the `dwFullSyncRateInmHz` attribute.

Finally the measurement rate of the overall EVDO measurement scan can be set, using the `dwMeasRateInmHz`. Note that this parameter specifies the frequency for a full scan of all the EVDO channels specified upfront. The more channels are specified, the less often they will be measured when the same measurement rate is used.

Once the settings have been done, the structure can be set in the ViCOM API using the `SetEvdoSettings()` method of the `CViComCdma2000Interface`.

Below a short example is shown that can be inserted into the CDMA 2000 sample at line 98. In that case, at least one frequency should be set to be scanned as EVDO frequency.

```
SEvdoControlSettings cEvdoSettings;
cEvdoSettings.dwMeasRateInmHz = 10000;
cEvdoSettings.dwFullSyncRateInmHz = 1000;
cEvdoSettings.dwShortSyncRateInmHz = 5000;
cEvdoSettings.dwShortSyncRangeInChips = 160;

EXEC_VICOM( "Configuring EvDO scanner",
            pcC2kScanner->SetEvdoSettings ( cError, cEvdoSettings ) );
```

7.6 EVDO BCH Demodulation

The BCH Demodulation part allows requests for CDMA and EVDO BCH Demodulation. The Message IDs are unique, therefore it is clear if CDMA or EVDO BCH Demodulation is requested. Of course, only if CDMA and EVDO channels are configured for scanning, the corresponding BCH Demodulation is available.

There are some general parameters for BCH Demodulation, the CPU Load and the Ec/Io Threshold. These parameters must be specified for CDMA and for EVDO, as appropriate.

The CPU Load for CDMA and EVDO BCH Demodulation together shall not exceed 50% of the full CPU Load utilized by ViCom. Therefore one ViCom instance can allocate from 10% up to 50% for CDMA or for EVDO BCH Demodulation only. If CDMA and EVDO BCH Demodulation is requested, only 10% up to 40% can be assigned to CDMA or EVDO and the other one is limited to the difference to 50%.

Please see section Demodulation Constraints of chapter 4 Programming with the ViCom Interface.

8 LTE Measurements

3rd generation network technologies like WCDMA and EvDO are the first step into solely IP based data exchange. The first real networks that are completely based on the transmission of IP packets are the LTE networks. The TSMW device can be used to perform LTE related measurements.



There is no support of LTE for the TSMx devices; this technology is only supported by the TSMW.

In the following sections, the programming API for the new TSMW and the LTE specific measurement details are described. Firstly, a short introduction into which and how LTE measurements are made in the ViCom is given.

After that the sample application coming with the ViCom delivery for LTE is explained. In the case of LTE, this sample application is a command line application similar to the small sample programs shown in the other ViCom APIs. A graphical sample application will be part of a future release.

8.1 Hardware

The hardware configuration is quite straightforward.

The TSMW is connected to a PC or Laptop via an Ethernet (LAN) connection on the rear Panel.

The TSMW has two frontends, each with a corresponding antenna connection on the rear panel (RF1 and RF2).

The user can also make use of the built in GPS receiver by connecting a GPS antenna to the TSMW using the connector on the rear panel.

These connections are shown on the figures below.

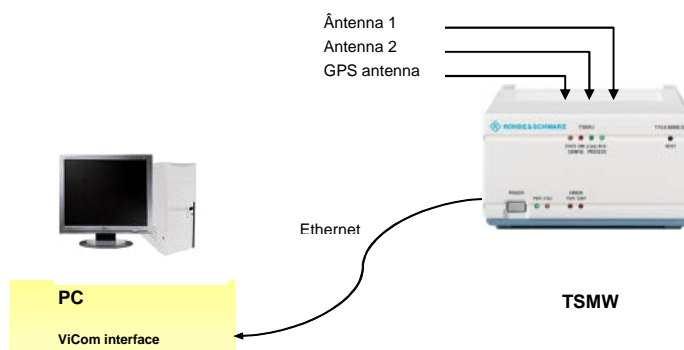


Figure 33 – Hardware setup



Figure 34 – Rear panel showing RF1, RF2, LAN and GPS connectors

The TSMW has the default IP address 192.168.0.2. To connect to the TSMW, a network connection on the PC should be configured with a suitable host IP address (e.g. 192.168.0.1).

After connecting the TSMW and configuring the host, the connection can be checked by opening a web-browser and entering the IP address of the TSMW. A HTML page will be displayed which shows details of the TSMW, the current firmware, etc.

8.2 Measuring LTE

Compared to GSM and WCDMA, LTE is a fairly new technology. Experience in the field of measurement modes and data analysis is currently gained, and the ViCom LTE module supports building such knowledge by providing different power value and channel impulse response measurements.

The LTE module itself offers a simple and straight forward interface to achieve that. In contrast to other interfaces, the configuration mainly concerns the way how the receiver synchronizes to the LTE signal and helps providing a fast synchronization when a-priori knowledge about the signal is available.

8.2.1 Configuration

The configuration of an LTE measurement mainly consists of a list of frequencies that shall be monitored. This frequency is the center frequency of the LTE band. For each frequency, additional information can be provided to improve the synchronization performance during the scanning.

One of that additional information concerns the number of OFDM symbols per slot used on that frequency. If that information is not available, the LTE scanner is able to automatically detect the number of symbols.

8.2.1.1 Receiver Frontend

The TSMW contains two frontends that can be used to perform measurements on different channels in parallel. These frontends are simply identified by their index, which is either 1 or 2. Each frequency can be measured on a specific frontend, which has to be specified during measurement setup (see also Resource Acquisition section on how to configure which front-end shall be used).

8.2.1.2 S-Sync To P-Sync Ratio

Furthermore, the power ratio between P-SYNC and S-SYNC can be specified, either as range or as a set of up to six single ratios. This information is used to provide fast synchronization, i.e. while the scanner is searching for signals it first uses the information provided in the power ratios to detect valid LTE data in the air. Even more important, a correct information places here increases the measurement accuracy significantly and reduced the probability of ghost code detection. So it is recommended that once the power ratios are known for the set of configured frequencies, that these values are in the measurement configuration.

In the case that a range is specified, the scanner has of course more possibilities to correlate to the detected signal, and must therefore spend more time in the signal processing algorithms. Providing up to six single ratios can boost the measurement performance and accuracy as described. Valid values for these ratios are returned as part of the measurement result and may be reused in further measurements.

The actual power ratios between the S-SYNC and P-SYNC are specified in dB. If, for example, the emitted power of the S-SYNC is 4 times higher than that used for the P-SYNC, the value 6 must be set in the `RatioList::afSSyncToPSyncRatioInDB` resp.

`RatioRange::fLowerRatioInDB/RatioRange::fUpperRatioInDB` of the `CViComLteInterfaceData::SFrequencySetting::SSyncToPSyncRatioSettings` structure. To calculate the value, the following formula can be used:

$$r[dB] = 10 \log \left(\frac{P_{S-Sync} [W]}{P_{P-Sync} [W]} \right)$$

The figure below visualizes shows some calculation examples for simple relations between S-SYNC and P-SYNC power.

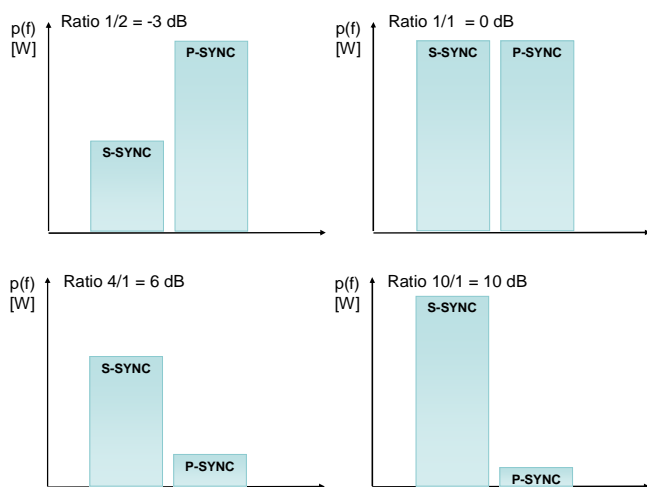


Figure 35 S-Sync to P-Sync ratios

Some standard values are listed in the table below. The S-Sync and P-Sync weighting refers to the power values in Watt. If the power values of S-Sync and P-Sync are given in dBm, the ratio is simply the difference between those values.

S-Sync	P-Sync	Value [dB]
10	1	10
5	1	6.99
4	1	6.02
2	1	3.01
1	1	0
1	2	-3.01
1	4	-6.02
1	5	-6.99
1	10	-10

8.2.1.3 Reference Signal Measurements

RSRP, RSRQ and RS-CINR sub band values can be measured with the LTE scanner as well. It measures those values on the six innermost resource blocks the scanner finds in a 100ms LTE measurement sample. According to the spec TS 36.214 it utilizes only antenna 0 and 1 for the RSRP and RSRQ measurements. The six innermost resource blocks will be located in the 1MHz bandwidth around the center frequency.

RSRQ value is calculated based on the measured RSRP value. This is done by putting the aforementioned RSRP in relation to the total in-band power of all the PBCH resource blocks within the 100ms signal sample.

To get those results, it is required to turn the RSRP measurement mode on in the measurement configuration. This is achieved by setting

`CViComLteInterfaceData::SFrequencySetting::wNarrowbandRefSignalMeasMode`

On top of the narrowband scanner a wideband measurement using the full channel bandwidth is available. This version allows extended reference signal measurements and supports RSRP, RSRQ, RS-CINR, RSSI as well as spectrum values. Since the measurements are done over a wider bandwidth they are intended to be slower and done on the stronger cells detected by the fast narrowband scanner. Compared to the narrowband mode this measurement needs much more IQ data (receiver time) and processing power (computation time). Rapid wideband measurements on all cells would lead to exhaustive resource usage. Therefore some conditions can be specified to limit the measurement to stronger cells only. These include:

1. Maximum number of cells to measure on the specified channel.
2. Minimum RSRP in dBm as reported by the narrowband scanner.
3. Minimum SSync-CINR in dB.
4. Maximum RSRP difference in dB to the strongest cell.

As the wideband measurement relies on sub band results the narrowband mode has to be activated using the above mentioned settings.

8.2.1.4 MIMO Measurements

Since ViCom 14.45 both TSMW receive antenna ports can be utilized for Multiple-Input Multiple-Output (MIMO) LTE measurements. All of the MIMO-specific measurements are based on the full bandwidth and can be applied to 4x2 and 2x2 MIMO systems. The output is based on the channel H-matrix with complex values, mean amplitude and phase. From the channel matrix, a singular-value decomposition is calculated. The result, the singular value, is used to obtain the condition number, which qualifies whether the channel is "ill-conditioned" (no MIMO applicable) or "well-conditioned" (MIMO usable).

As a requirement for MIMO one needs to activate the narrow- and wideband RS-CINR measurements. Intentionally the MIMO measurements are only to be done for strong cells, which means there are switches to restrict the calculations. In the same way it is done for wideband RS-CINR.

The basic step to activate MIMO is to set the

`ViComLteInterfaceData::SFrequencySetting::SMimoSettings::wMimoMeasMode` parameter to either `wMIMO_MODE_2x2` for 2x2 MIMO configurations or to `wMIMO_MODE_2x4` for 2x4 MIMO respectively.

The measurements rely on the H-channel matrix, which is calculated per cell and per resource block. This enables further drill-down to determine factors such as interference, multipath fading, antenna correlation and noise in relation to their spectral position

8.2.2 Measurement Result

Once the measurement has been set up, the measured power values are returned when requested using `GetResult()`, as shown in the other ViCom applications. There are up to two different result structures returned in the measurement mode: The channel impulse responses and the power values combined with the related CINR values of the S-SYNC channel.

8.2.2.1 Channel Impulse Responses

The channel impulse responses are always calculated for a 100ms block. The signal processing performed on that timeframe may deliver any combination of the following two results, if the signal is good enough:

CIR Peaks

From a 100ms S-Sync signal the processing algorithms in the LTE Scanner module extract the CIR peaks. The CIR Peaks also come with Doppler frequency estimates, which are a result from the way the values are calculated internally.

Power Delay Profiles

Based on that CIR Peaks, which are not necessarily equally distributed on a time axis, the power delay profile is derived. The power delay profile consists of a list of power values which are actually sampled from the CIR Peak value resp. some interpolated data between those peaks. The sampling rate

`SCir::SPowerDelayProfile::fSamplingTimeInSec` is constant for all the values, but it can happen that invalid power values are a part of the result when there is no meaningful interpolation result available (when peaks are missing, for example).

One result may be a so-called Power-Delay-Profile. Power values are measured at a constant sampling rate, which is returned in. The single power value results measured at that rate are specified in the

`SCir::SPowerDelayProfile::SPowerDelayProfileValues::psValuesInDBm100` array.

The original peaks are distributed in a much higher resolution than the power delay profile values. In the figure below the connection between the CIR Peaks and the sampled power delay profile values is depicted. Please note however that this is a schematic figure, which shall clarify the idea behind the two different result types.

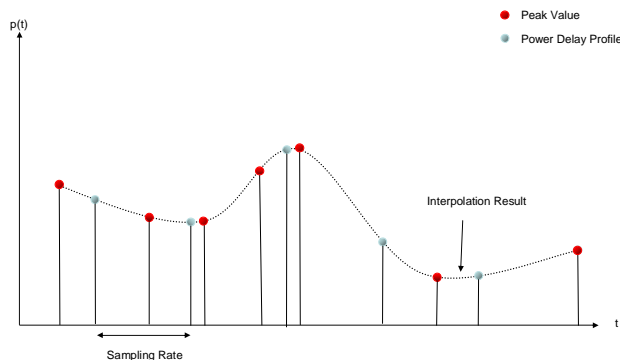


Figure 36 CIR Peak and Power Delay Profile

8.2.2.2 Channel Power & CINR Values

Another result lists the power measured for the S-Sync channels in the 100ms signal. The related power values of the P-Sync channel can be derived using the reported S-Sync to P-Sync ratio by simply adding that value to the S-Sync power.

The power for the S-Sync channel can either be calculated from a complete OFDM block or a single OFDM symbol. In the latter case, the time delay between the start of the block and the symbol that has been used to derive the power is listed in the result. Besides the power value of the channel, a set of CINR values is calculated. If possible, the CINR is calculated for both P-Sync and S-Sync channels separately, but in case that this is not possible, an averaged CINR over both channels is returned.

Additionally, an amplitude CINR is calculated for each of the two channels, where the phase CINR component is removed from the overall CINR. If the difference between the CINR and the amplitude CINR is high, this indicates a phase noise problem in the sender or receiver, and that there is no real interferer that causes the situation.

8.2.3 Error Handling

The LTE ViCom interface is the first that supports C++ exceptions natively. For every method in the interface (and also in the basic interface) taking a `CViComError` reference to be filled with potential error messages, there is a counterpart method that will throw a `CViComError` object in case that a problem arises. All the remaining behavior is consistent between both methods, so you can choose which version fits your needs best.

8.3 BCH Demodulation

In the same way as offered by the other ViCom interfaces, it is possible to perform BCH demodulation in the LTE ViCom interface as well. The main difference here is that it is possible to define different demodulation load on a front-end for different channels, whereas the other interfaces only offer the possibility to define a global demodulation load.

This minimum load limit that is spent on demodulation is combined with the general load indicator. A detailed description how to use the BCH demodulation capabilities is given in the chapter "Using the Demodulators" on page 33.

8.4 GUI Sample Application

The ViCom installer comes with a GUI sample application to demonstrate how to make LTE measurements through the ViCom interface. The complete source code of the sample application is inside the “SampleForViComLTE” directory of the ViCom installation directory. It is possible to build both the debug and release sample applications by opening SampleForViComLte.vcproj with Visual C++ 2008 and then performing a batch build. The binaries will be generated inside the “Application” directory with the name SampleForViComLTERelease.exe for the release build and SampleForViComLTEDebug.exe for the debug one. The release binary also comes directly with the ViCom installation and is used as the reference of discussion in this document. The following topics will be covered in this chapter:

- Using the Sample Application
- Code analysis

8.4.1 Using the Sample Application

The interface of the sample application looks like this

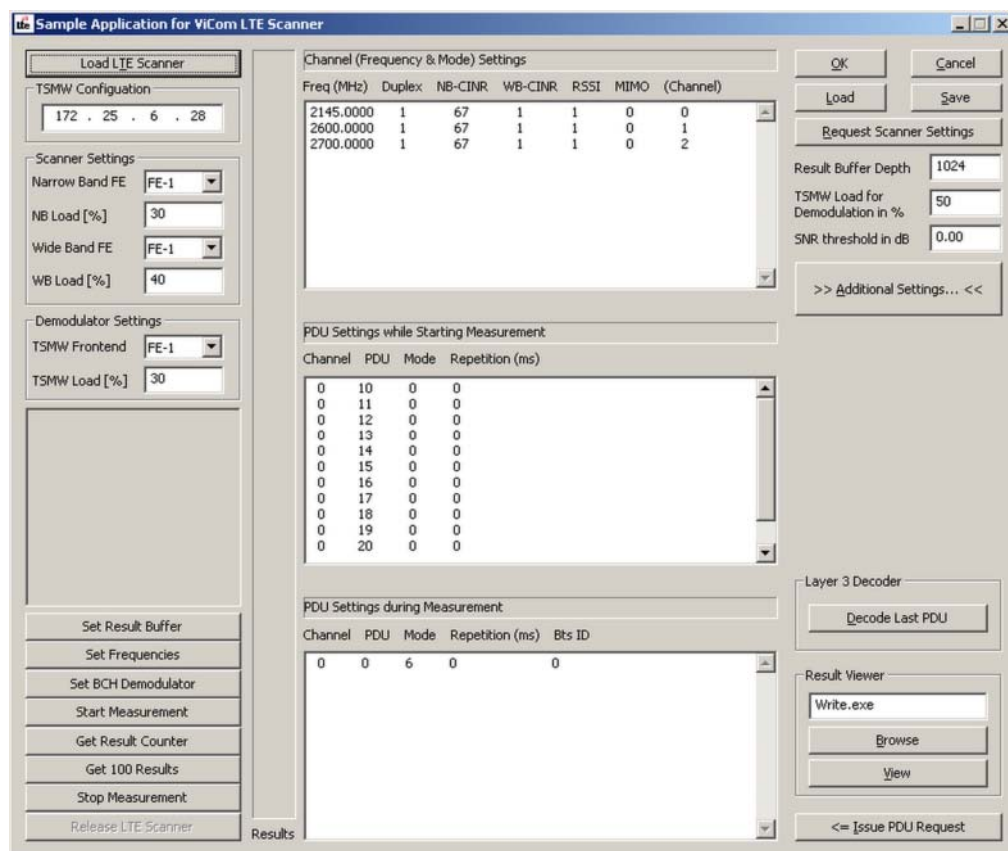
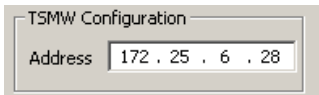


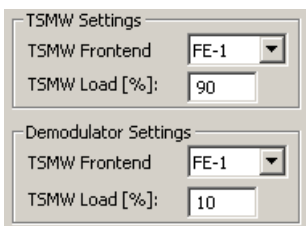
Figure 37 LTE Sample Application

8.4.1.1 Connecting to TSMW

1. First define the IP address of the TSMW



2. Specify the load assignments for the front ends.

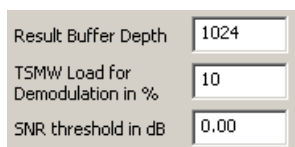


3. Press the "Load LTE Scanner" button at the upper left corner of the sample application. The application will try to connect to the TSMW at the specified IP address and assign the front ends load accordingly. If the connection is successfully established, some information about the TSMW will be displayed in the read-only text box at the middle left of the dialog box.



8.4.1.2 Configure the ViCom interface

At the right side of the dialog box, there are three text boxes to configure some parameters of the scanning.



“Result Buffer Depth”

This Setting configures the maximum number of LTE scan results ViCom will buffer. The results are buffered after the measurement is started, and if they are not retrieved by any GetResult() function calls, and when the number of buffered results reach the number specified in this text box, the earliest results will be discarded. The default value is 1024. Pressing the “Set Result Buffer” button will set this value for ViCom.

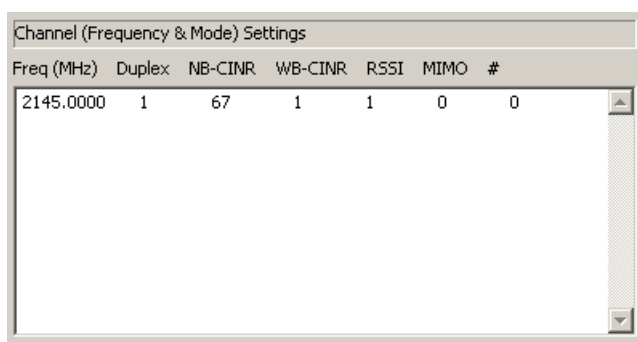
“TSMW Load for Demodulation in %”

This setting specifies the load percentage for the demodulator. The higher this value is, the more resource TSMW will spend on demodulation. The maximum value can be set is 50. Pressing the “Set BCH demodulator” button will set this value for ViCom.

“SNR threshold for Demodulation in dB”

This setting specifies the SNR threshold for the demodulator. Only when the calculated SNR of the signal is higher than the value specified here, the demodulator will demodulate the signal. The default value is 0 (dB). Pressing the “Set BCH demodulator” button will set this value for ViCom.

Frequency / Channel Settings



This text box specifies the frequency / channel settings. Each line configures one channel and consists of seven numbers. The meaning of these parameters are shown in [Table 5 Frequency / Channel Settings Parameters](#). Press “Enter” to create a new line in the text box and configure another channel (The channel number).

Table 5 Frequency / Channel Settings Parameters

Seq.	Label	Meaning
1	Freq (MHz)	Channel Frequency in MHz.
2	Duplex	Duplex Mode. (1 – FDD; 2 - TDD)
3	NB-CINR	Narrow-Band RS-CINR Mode. This is a bit flag, please refer to SFrequencySetting::wNarrowbandRefSignalMeasMode inside “ViComLteInterfaceData.h” for more information. The default value is 67.
4	WB-CINR	Wide-Band RS-CINR Mode. Set to ‘1’ to enable wide-band RS-CINR measurement, set to ‘0’ to disable it.
5	RSSI	Reference Signal Measurement modes. The default is ‘1’ for RSSI Normal Mode.
6	MIMO	MIMO mode. (0 – No MIMO measurement; 1 – 2 x 2 MIMO measurement; 2 – 4 x 4 MIMO measurement)
7	#	Channel Number. This number is automatically generated from the program, and it is the same as the zero-based line number of the edit box. This value is for the user’s reference only, changing it at the edit box won’t affect ViCom’s frequency settings.

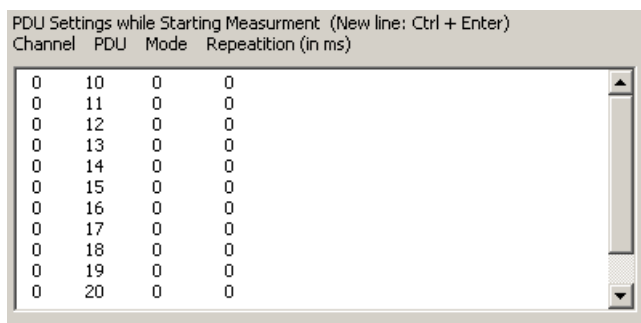
Demodulation mode

There are different modes to perform LTE demodulations at measurement startup and during measurement. The meaning of each demodulation mode and its usage is explained in the following table.

Table 6 LTE Demodulation Modes

Mode (Integer value)	Meaning	Usage ¹
PDU_DEMOD_ONCE (0)	Demodulate the PDU for each BTS at the specified channel only once.	S
PDU_DEMOD_ON_CMD (1)	The PDU is only demodulated if an extra command for demodulation requests the demodulation.	S
PDU_DEMOD_REPETITION (2)	After each successful demodulation a new one will be started automatically after the repetition time specified in wRepetitionDelayIn100ms.	S
PDU_DEMOD_CHANNEL_RESET (6)	Delete all demodulation results for all BTS at the specified channel. Restart demodulation for this channel as specified in the command.	D

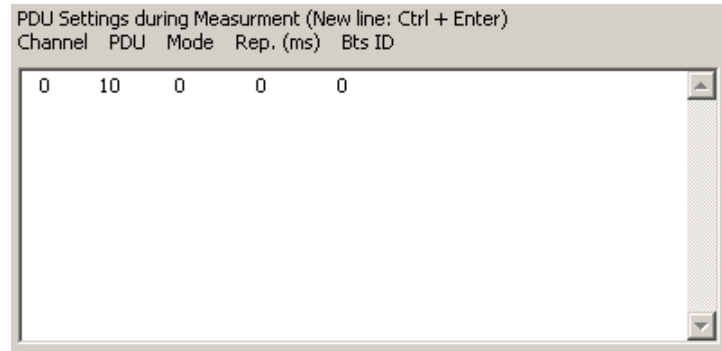
Start Measurement



This text box specifies demodulation configurations for each channel when starting measurement. Each line configures the parameters for one PDU/demodulation mode and consists of four numbers. The first one specifies the channel to configure the demodulation parameters on, it must be one of the channel number specified in the Frequency / Channel Settings (the first multi-line) text box; The second one specifies which PDU to demodulate (for example: 10 for MIB, 13 for SIB 3 and so on); The third one specifies the demodulation mode as described in [Table 6 LTE Demodulation Modes](#); The last one specifies the time if the demodulation mode is PDU_DEMOD_REPETITION (It should be zero for all other cases). Press “Ctrl + Enter” to create a new line in the text box and configure another demodulation request.

¹ S: Starting Measurement; D: During Measurement.

During Measurement



This text box specifies the demodulation requests to be sent during the measurement. Each line specifies one request and consists of five numbers. The first four parameter is the same as described in the previous paragraph Start Measurement. The fifth one specifies the BTS. It shall be set to zero by default. Press “Ctrl + Enter” to create a new line in the text box and configure another demodulation request.

If all the demodulation requests are properly configured, you can then press the “Issue Request button” to issue these demodulation requests while measurement is running, i.e. after the Start Measurement button is pressed.

Saving / Loading

The buttons at the upper right corner are for setting saving/loading (serialization). Pressing the “OK” button will save the configuration to the default configuration file “LteScannerSettings.bin” (which will be examined and loaded on program start up) and dismiss the dialog box, while pressing the “Cancel” button will dismiss the dialog without saving the configurations. The “Load” / “Save” buttons are to load /save the configurations from/to a file whose path is chosen at the user’s preference.

Pressing the button “Request Scanner Settings” will refresh the text boxes with the current settings of the ViCom instance.

Additional Channel Settings

Pressing the button “Additional Settings” will open the “Additional Channel Settings” dialog box. This dialog box offers a lot of additional settings for the channels.

The dialog box can be used to configure one single channel or all channels.

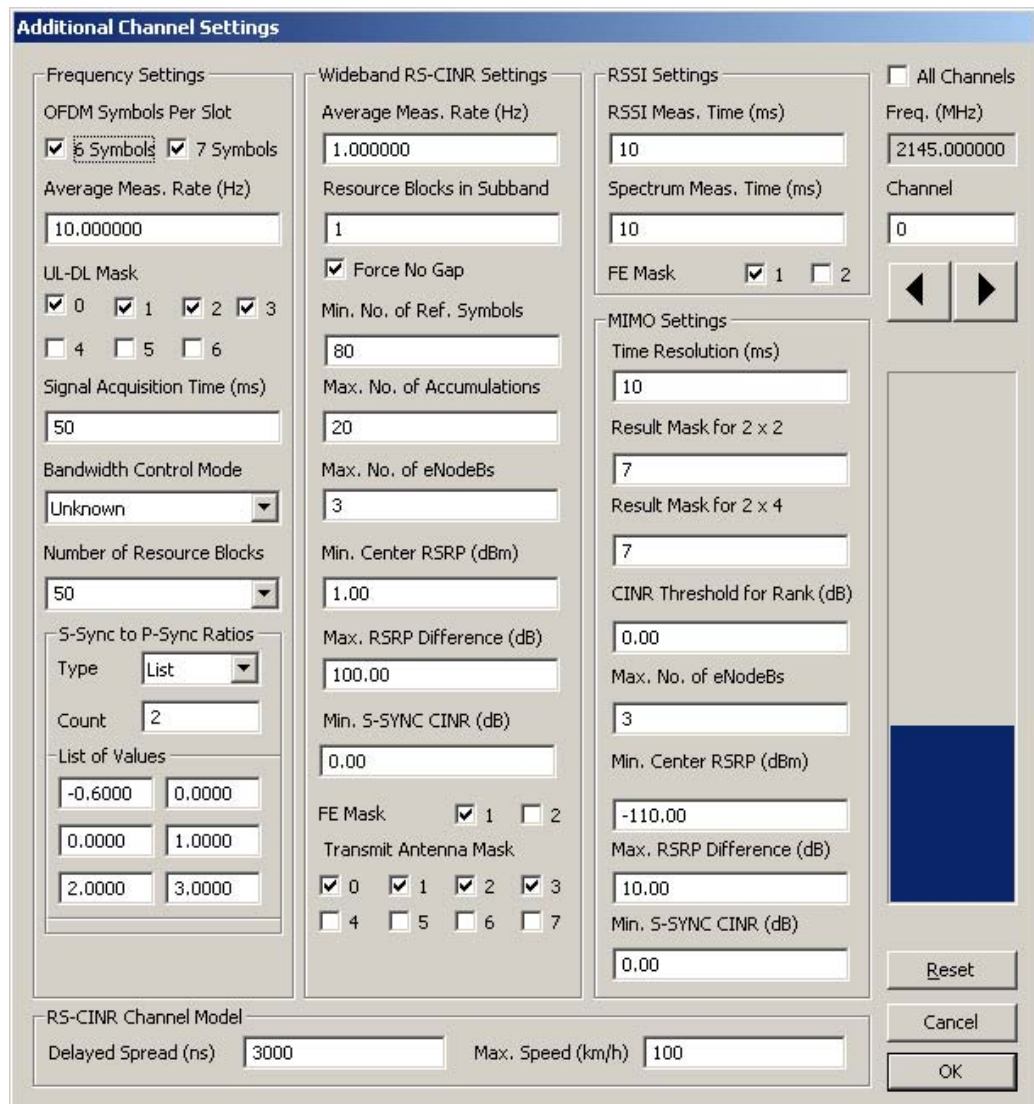


Figure 38 Additional Channel Settings for Single Channel

Leave the “All Channels” check box at the right upper corner of the dialog box unchecked to configure a single channel. In this single channel configuration mode, you can switch among the channels by clicking the big horizontal spin control at the right side of the dialog box, or you can directly input the channel number into the edit box labeled “Channel”, just above the spin control. The large vertical horizontal progress bar is an obvious indicator of the channel you are configuring, it changes it’s progress position when the channel is changed. When you made some changes in the dialog box, you will be prompted to save the changes to that channel before switching to another channel.

Figure 39 Additional Channel Settings for All Channels

Tick the “All Channels” checkbox to configure all channels. In this mode, if a specific setting is different among the channels, the value will be displayed as blank (for text boxes or combo boxes) or intermediate-checked (for checkboxes), and if you leave these blanks or intermediate-checks unchanged, the dialog will not modify these settings when you click OK to save the changes.

8.4.1.3 Start Measurement

After all the parameters are properly configured, you can press the “Start Measurement” button to start measurement and ViCom will retrieve and buffer scan / demodulation results from the TSMW. To get these the scan results, press the “Get 100 Results” button. It will gather 100 scan results and write them to the log file “LogFiles\ViComLteMeasurements.txt” every time it is pressed.

8.4.1.4 Stop Measurement

To end measurement, press the “Stop Measurement” button to stop the measurement and the “Release LTE Scanner” button to destroy the ViCom interface.

8.4.2 Code analysis

8.4.2.1 Initialization

The initialization is done in the

`CSampleForViComLteDlg::OnBnClickedLoadScanner()` function, the related snippets are shown below:

```
bLoaded= m_ViComLoaderW.ConnectTsmw(err, ipAddressString);
if (bLoaded) {
    m_pViComInterface = m_ViComLoaderW.GetpInterface();
    m_state = InterfaceCreated;
    UpdateControlStates();
    // reset resource handles
    m_hNarrowBand      =
CViComBasicInterfaceData::InvalidDeviceHandle();
    m_hWideBand       =
CViComBasicInterfaceData::InvalidDeviceHandle();
    m_hDemodulatorResource =
CViComBasicInterfaceData::InvalidDeviceHandle();
}
```

The `ConnectTsmw()` function tries to establish a connection to the TSMW, and the `GetpInterface()` function returns a pointer to the ViCom interface if the connection is successful. From there onwards, the sample application call all the ViCom functions through this interface.

8.4.2.2 Configurations

The configurations of different parameters in the sample application is achieved by calling different configuration functions in the ViCom interface that can be easily understood by reading the code inside the button-clicked functions.

8.4.2.3 Measurement

The measurement is started by calling the

`GetBasicInterface().StartMeasurement()` function through the ViCom interface. Once this function is called, ViCom will start receiving and buffer measurement results from the TSMW it connected to. To get these measurement results, call `GetResult()` over the ViCom interface.

L3 result

To get L3 decode result, the sample application needs first to check the `pPduResult` of the `SMeasResult` structure returned by the `GetResult()` function. If it is not `NULL`, it contains some PDU data, and to decode these PDU data through L3 decoder, the sample application fills up a `SL3DecoderRequestData` structure with contents copied from the `pPduResult`, and then call the `RetrieveTextForPDU()` function to retrieve decoded result from the Layer 3 Decoder.

8.4.2.4 Stop measurement

To stop a measurement, simply call the `GetBasicInterface().StopMeasurement()` function through the ViCom interface. Calling `DisconnectTsmw()` will terminate the connection and destroy the ViCom interface.

8.5 Command Line Sample Application

The LTE ViCom API comes with a simple command line application that demonstrates the usage of the API. It can be found in the ViCom installation folder in the directory called "SampleForViComLTE_cmd".



The sample project has been made for Visual Studio 2008. The other graphical sample applications have been made utilizing Visual Studio .NET 2003.

This command line sample application is structured in the same way as the Code Examples in the other ViCom API (refer to the other chapters describing technologies like GSM, WCDMA etc). It mainly starts a measurement and prints the signal details to the console, until no results are detected for 5 seconds resp. a key is pressed.

However, there are some notable differences to the other console examples provided in this manual.

First of all, the exception throwing counterparts of the ViCom interface functions are used, so there is no need for the `EXEC_VICOM` macro found in the other samples. This improves the overall readability of the code and simplifies dealing with the API.

The next difference concerns the way the device is loaded, which is natural since the TSMW is used to perform those measurements. The IP address is set in the code to the default value, if your device has been configured differently you can change that in the `_tmain()` function.

Another (minor) differences are that the code is structured in a different way, i.e. it does not put all the logic into one big function, and that typedefs are used to reduce the number of nested references to the result structures.

```
#include "stdafx.h"
#include "ViComLteInterface.h"
#include "ViComLoader.h"

5 #include <conio.h>
```

```

#include <math.h>

typedef CViComBasicInterfaceData::DeviceResourceHandle
ResourceHandle;
10 typedef CViComLteInterfaceData::SFrequencySetting
SFrequencySetting;
typedef CViComLteInterfaceData::SChannelSettings
SChannelSettings;
15 typedef CViComLteInterfaceData::SBchDemodulationSettings
SBchDemodulationSettings;
typedef CViComLteInterfaceData::S_PDU_Requests::S_LTE_PDU_Request
S_LTE_PDU_Request;
typedef CViComLteInterfaceData::S_LTE_RATIO_CONFIDENCE_INTERVALL
S_LTE_RATIO_CONFIDENCE_INTERVALL;
20 typedef CViComLteInterfaceData::S_LTE_RS_CINR_VALUE
S_LTE_RS_CINR_VALUE;
typedef CViComLteInterfaceData::FrameStructureType
FrameStructureType;
typedef CViComLteInterfaceData::SSettings SSettings;
25 typedef CViComLteInterfaceData::SMeasResult SMeasResult;
typedef WORD
RefSignalMeasMode;

typedef SMeasResult::SPduResult SPduResult;
30 typedef SMeasResult::SSignals SSignals;
typedef SMeasResult::SSignals::SCir SCir;
typedef SMeasResult::SSignals::SCir::SPowerDelayProfile
SPowerDelayProfile;
35 typedef SMeasResult::SSignals::SCir::SPeak SPeak;
typedef SMeasResult::SSignals::SPowerValue SPowerValue;
typedef SMeasResult::SWidebandRsCinrResult
SWidebandRsCinrResult;
typedef SMeasResult::SRssiAndSpectrumResult
SRssiAndSpectrumResult;
40 typedef SMeasResult::SMimoResult SMimoResult;

using namespace std;

/**
45 Create the measurement configuration for the test scenario.
*/
SSettings getLteMeasConfig( ResourceHandle resourceNarrowbandScanner,
ResourceHandle resourceWidebandScanner, ResourceHandle resourceDemodulator )
{
50 SSettings settings = SSettings();
settings.ResultBufferDepth.dwValue = settings.ResultBufferDepth.dwMax;
double dFrequenciesInMhz[] = {796.0};
DWORD dwNumberOfChannels = sizeof( dFrequenciesInMhz ) / sizeof( double );

55 // Activate BCH demodulation
DWORD dwRequests = 12 *
dwNumberOfChannels; // Request 12 SIBs
SBchDemodulationSettings& rBCH =
60 settings.BchDemodulationSettings;
rBCH.sSINRThresholdDB100 =
rBCH.sMinSINRThresholdDB100;
rBCH.wLoadInPercent =
rBCH.wOptLoadInPercent;
rBCH.sStartMeasurementRequests.dwCountOfPDURRequests = dwRequests;
65 rBCH.sStartMeasurementRequests.pPDURRequest = new
S_LTE_PDU_Request[dwRequests];
memset( rBCH.sStartMeasurementRequests.pPDURRequest, 0,
rBCH.sStartMeasurementRequests.dwCountOfPDURRequests * sizeof(S_LTE_PDU_Request)
);
70 // Configure channels
for( DWORD dwChannelIndex = 0; dwChannelIndex < dwNumberOfChannels;
++dwChannelIndex )
75 {
settings.ChannelSettings.dwCount = dwChannelIndex+1;
}
}

```

```

        SFrequencySetting& s =
settings.ChannelSettings.aTableOfFrequencySetting[dwChannelIndex];

80     //
        // Select the resources used to scan this channel.
        //
        s.resourceHandleForNarrowbandScanner = resourceNarrowbandScanner;

85     //
        // Select the resources used to scan this channel.
        //
        s.resourceHandleForWidebandScanner = resourceWidebandScanner;

90     //
        // Select the resources used to demodulate this channel.
        //
        s.resourceHandleForDemodulator = resourceDemodulator;

95     //
        // The frequency that shall be measured, in Hertz and as floating
        // point number
        //
        s.dCenterFrequencyInHz = dFrequenciesInMhz[dwChannelIndex] * 1000 * 1000;

100    //
        // Since we do not know here how many symbols shall be used,
        // the default setting is used which slows down synchronization
        // a little bit, so if that information is known upfront,
        // change the line below accordingly.
105    //
        s.dwSymbolsPerSlotMask = s.dwDefaultSymbolsPerSlot;

        //
        // Type of the channel (FDD or TDD) must be known upfront to
        // make the scanner handle the power management correctly.
        //
110    s.enFrameStructureType = FrameStructureType::FDD;

        //
        // Define the measurement rate, i.e. how many blocks of 100ms
        // shall be measured per second. Note that this is not a
        // strong constraints that will be fulfilled in every second,
        // but the scheduler attempts to create such a measurement
        // rate over a longer period of time.
115    //
        // A value of 10 is the theoretical maximum value per receiver
        // in the TSMW, as is easy to understand. The second
        // TSMW receiver will be supported, but this will
        // come in a future version.
120    //
        s.dAvgBlockCountPer1000Sec = s.dwMaxAvgBlockCountPer1000Sec;

        //
        // Setup Reference Signal measurement mode for RSRP/Q / CINR measurement.
        // Narrowband RS-CINR is required for the wideband RS-CINR!
        //
125    s.wNarrowbandRefSignalMeasMode =
SFrequencySetting::wNARROWBAND_RSRP_RSRQ |

130    SFrequencySetting::wCENTER_RSCINR_6x180KHZ |

        SFrequencySetting::w1MHZ_FILTER_NOISE_FOR_15RB;

        s.enBandwidthCtrlMode =
140    SFrequencySetting::BandwidthCtrlMode::BW_FROM_MIB_ONCE; // Needs active MIB
        demodulation!

        s.WidebandRsCinrSettings.wWidebandRsCinrMeasMode =
SFrequencySetting::SWidebandRsCinrSettings::wWIDEBAND_RS_CINR;
145    s.WidebandRsCinrSettings.dwFrontEndSelectionMask =
CViComBasicInterfaceData::TSMW_RF_FRONTEND_1;

```

```

150 //
// If the power ratio between P-SYNC and S-SYNC is known upfront,
// maybe from previous measurements, then the measurement accuracy
// can be increased (which reduces number of reported ghost codes).
// In this sample, the default is to tell the scanner to expect
// the ratio between the S-SYNC and P-SYNC to be between 1/4 and
155 // 1/1. The ratios are given in dB, which is the results of
// 10 * log ( P(S-SYNC)/P(P-SYNC) ).
//
s.enSSyncToPSyncRatioType = s.SSyncToPSyncRatioType::RatioRange;
s.SSyncToPSyncRatio.RatioRange.fLowerRatioInDB = -6.f;
160 s.SSyncToPSyncRatio.RatioRange.fUpperRatioInDB = 0.f;

//
// Setup RSSI and spectrum measurements
//
165 s.RssiSettings.wRssiMeasMode =
CViComLteInterfaceData::SFrequencySetting::SRssiSettings::wRSSI_NORMAL_MODE;
s.RssiSettings.wRssiMeasTimeInMs = 10;
s.RssiSettings.wSpectrumMeasTimeInMs = 10;

s.RssiSettings.dwFrontEndSelectionMask =
170 CViComBasicInterfaceData::TSMW_RF_FRONTEND_1;

//
// Setup MIMO measurements
//
175 s.MimoSettings.wMimoMeasMode = 0;

//
// As an alternative, a list of power ratios can be specified. This helps
// increasing the scanning accuracy and also provides faster
180 synchronization.
// The code below shows how two ratios can be set in that case. Note that
// although that looks like the code shown above, it has a different
meaning:
// The configuration below only checks the two ratios -6 dB and 0 dB,
185 whereas
// the one above also tries ratios in between that two borders.
//
//s.enSSyncToPSyncRatioType = s.SSyncToPSyncRatioType::RatioList;
//s.SSyncToPSyncRatio.RatioList.dwValueCount = 2;
190 //s.SSyncToPSyncRatio.RatioList.afSSyncToPSyncRatioInDB[0] = -6.f;
//s.SSyncToPSyncRatio.RatioList.afSSyncToPSyncRatioInDB[1] = 0;

// Configure BCH-Demodulation
DWORD dwRequestsPerChannel = 12;
DWORD dwRequestStartIndex = dwChannelIndex * dwRequestsPerChannel;
S_LTE_PDU_Request& rRequestMIB =
rBCH.sStartMeasurementRequests.pPDURequest[dwRequestStartIndex];
S_LTE_PDU_Request& rRequestSIB =
200 rBCH.sStartMeasurementRequests.pPDURequest[dwRequestStartIndex +1];

// Request MIB
rRequestMIB.dwChannelIndex = dwChannelIndex;
rRequestMIB.ePDU =
205 CViComLteInterfaceData::LTE_PDU_FOR_MIB;
rRequestMIB.eDemodulationMode =
CViComLteInterfaceData::PDU_DEMOD_ONCE;
rRequestMIB.wRepetitionTimeOutInMs = 0;
rRequestMIB.dwBtsId = 0;
// Request SIB 1
210 rRequestSIB.dwChannelIndex = dwChannelIndex;
rRequestSIB.ePDU =
CViComLteInterfaceData::LTE_PDU_FOR_SIB1;
rRequestSIB.eDemodulationMode =
CViComLteInterfaceData::PDU_DEMOD_ONCE;
215 rRequestSIB.wRepetitionTimeOutInMs = 0;
rRequestSIB.dwBtsId = 0;

```

```

    // Request SIB 2 .. SIB 11
    for ( DWORD idx = 2; idx < dwRequestsPerChannel; idx++ )
220     {
        S_LTE_PDU_Request&   rRequestPDU =
rBCH.sStartMeasurementRequests.pPDURequest[dwRequestStartIndex + idx];
        rRequestPDU.dwChannelIndex       = dwChannelIndex;
        rRequestPDU.ePDU                 =
225     CViComLteInterfaceData::eLTE_PDU(idx+10); // Start with LTE_PDU_FOR_SIB2
        rRequestPDU.eDemodulationMode    =
CViComLteInterfaceData::PDU_DEMOD_ONCE;
        rRequestPDU.wRepetitionTimeOutInMs = 0;
230     rRequestPDU.dwBtsId                = 0;
    }
}

return settings;
235 }

/**
240 Helper operator to write information onto console.
You can extend this to add additional output when you are interested in more
details on the selected signals.
*/
ostream& operator<< (ostream& os, const SMeasResult& res)
245 {
    os << "dwChannelIndex = " << res.dwChannelIndex << endl;
    os << "dwTimestampInMs = " << res.dwPcTimeStampInMs << endl;

    int j=0;
    for (SViComList<SSignals>::SLinkedObject* p = res.ListOfSignals.pFirst;
250     p!=NULL;
        p=p->pNext, ++j)
    {
        //
255     // General result describing the detected LTE station
        //
        os << "\tPhysicalCellId [" << j << "] = " << p->wPhysicalCellId << endl;
        os << "\tScanner ID = " << p->dwScannerBtsIdent << endl;
        os << "\tNumber of Symbols per Slot = " << unsigned(p-
>bNumberOfSymbolsPerSlot) << endl;
260     os << "\tFrame Structure Type = " << (p->enFrameStructureType ==
FrameStructureType::FDD ? "FDD" : "TDD") << endl;

        if ( p->pfSSyncToPSyncRatioInDB )
265     os << "\tS-Sync To P-Sync Ratio [dB] = " << *p->pfSSyncToPSyncRatioInDB
<< endl;

        os << "\tExpected Time Drift [nsec/sec] = " << p-
>fExpectedTimeDriftInNsPerSec << endl;
270     os << "\tSigma Time Drift [nsec/sec] = " << p->fSigmaTimeDriftInNsPerSec
<< endl;

        if ( 0 < p->sRefSignal.bAntennaMaskUsedForRSRP )
275     {
        os << "\tbAntennaMaskUsedForRSRP = " << p-
>sRefSignal.bAntennaMaskUsedForRSRP << endl;
        os << "\tsCenterRSRPinDBm100 = " << p->sRefSignal.sCenterRSRPinDBm100
<< endl;
        os << "\tsPBCHbasedRSRQinDB100 = " << p-
280     >sRefSignal.sPBCHbasedRSRQinDB100 << endl;
        os << "\tbNumberOfSymbolsUsedForRSRQ = " <<(unsigned int) p-
>sRefSignal.bNumberOfSymbolsUsedForRSRQ << endl;
    }
    if ( 0 < p->sRefSignal.pbUpDownLinkConfig )
285     os << "\tbUpDownLinkConfig = " << (unsigned int) *p-
>sRefSignal.pbUpDownLinkConfig << endl;
    if( 0 < p->sRefSignal.pNarrowbandRsCinrValues )
    {

```

```

290         os << "\tSubband RS-CINR result:" << endl;
        for( DWORD i = 0; i < 4; i++)
            os << "\t\tbRsCinrMeasResultConfig[" << i << "] = " << p-
>sRefSignal.bRsCinrMeasResultConfig[i] << endl;
            for( DWORD i = 0; i < p->sRefSignal.dwCountOfNarrowbandRsCinrValues;
i++ )
295         {
            os << "\t\t[" << i+1 << "]: Linear average RE power [dBm] = ";
            if( p-
>sRefSignal.pNarrowbandRsCinrValues[i].sLinAverageREpowerInDBm100 ==
S_LTE_RS_CINR_VALUE::sInvalidValue )
300             os << "-" << endl;
            else
                os << 0.01 * p-
>sRefSignal.pNarrowbandRsCinrValues[i].sLinAverageREpowerInDBm100 << endl;
            os << "\t\t\t RS-CINR [dB] = ";
305             if ( p->sRefSignal.pNarrowbandRsCinrValues[i].sRsCinrInDB100 ==
S_LTE_RS_CINR_VALUE::sInvalidValue )
                os << "-" << endl;
            else
                {
310                 os << 0.01 * p-
>sRefSignal.pNarrowbandRsCinrValues[i].sRsCinrInDB100 << endl;
                os << "\t\t\t\t\t68% confidence interval [dB]: ";
                if ( p-
>sRefSignal.pNarrowbandRsCinrValues[i].ConfidenceInterval68PercentOfsRsCinrInDB1
315 00.bDeviationToLowerRatioInDB10 ==
S_LTE_RATIO_CONFIDENCE_INTERVAL::bInvalidValue )
                    os << "-" << endl;
                else
                    os << "To lower ratio: " << 0.1 * p-
320 >sRefSignal.pNarrowbandRsCinrValues[i].ConfidenceInterval68PercentOfsRsCinrInDB1
00.bDeviationToLowerRatioInDB10 << " To higher ratio: ";
                    if ( p-
>sRefSignal.pNarrowbandRsCinrValues[i].ConfidenceInterval68PercentOfsRsCinrInDB1
325 00.bDeviationToHigherRatioInDB10 ==
S_LTE_RATIO_CONFIDENCE_INTERVAL::bInvalidValue )
                        os << "-" << endl;
                    else
                        os << 0.1 * p-
>sRefSignal.pNarrowbandRsCinrValues[i].ConfidenceInterval68PercentOfsRsCinrInDB1
330 00.bDeviationToHigherRatioInDB10 << endl;
                        os << "\t\t\t\t\t95% confidence interval [dB]: ";
                        if ( p-
>sRefSignal.pNarrowbandRsCinrValues[i].ConfidenceInterval95PercentOfsRsCinrInDB1
335 00.bDeviationToLowerRatioInDB10 ==
S_LTE_RATIO_CONFIDENCE_INTERVAL::bInvalidValue )
                            os << "-" << endl;
                        else
                            os << "To lower ratio: " << 0.1 * p-
340 >sRefSignal.pNarrowbandRsCinrValues[i].ConfidenceInterval95PercentOfsRsCinrInDB1
00.bDeviationToLowerRatioInDB10 << " To higher ratio: ";
                            if ( p-
>sRefSignal.pNarrowbandRsCinrValues[i].ConfidenceInterval95PercentOfsRsCinrInDB1
345 00.bDeviationToHigherRatioInDB10 ==
S_LTE_RATIO_CONFIDENCE_INTERVAL::bInvalidValue )
                                os << "-" << endl;
                            else
                                os << 0.1 * p-
>sRefSignal.pNarrowbandRsCinrValues[i].ConfidenceInterval95PercentOfsRsCinrInDB1
350 00.bDeviationToHigherRatioInDB10 << endl;
                                }
                            }
                    }
                }
            else
355         {
            os << "\tNo subband RS-CINR measurement" << endl;
        }

        //
        // CIR Result

```

```

360     //
        if ( p->pCir )
        {
            os << endl << "\tChannel Impulse Response" << endl;
            os << "\t\tPC Timestamp = " << p->pCir->dwPcTimeStampInMs << endl;
365     os << "\t\tDevice Time [ns] = " << p->pCir->u64DeviceTimeInNs<< endl;

            if ( p->pCir->pPowerDelayProfile )
            {
370                 const SPowerDelayProfile* pdp = p->pCir->pPowerDelayProfile;

                    os << "\t\t\tInband Power [dBm] = " << pdp->fInbandPowerInDBm <<
endl;
                    os << "\t\t\tTotal Aggregated Power [dBm] = " << pdp->
>fAggregatePowerInDBm << endl;
375                 os << "\t\t\tNoise Floor [dBm] = " << pdp->fNoiseFloorInDBm << endl;
                    os << "\t\t\tSampling Time [sec] = " << pdp->fSamplingTimeInSec <<
endl;
                    os << "\t\t\tPower Values = ";
                    for ( unsigned int valIdx = 0; valIdx<pdp->
380 >PowerDelayProfileValues.dwCountOfValues; ++valIdx )
                        {
                            if ( pdp->PowerDelayProfileValues.psValuesInDBm100[valIdx] ==
SPowerDelayProfile::SPowerDelayProfileValues::sInvalidValue )
                                os << "- ";
385                             else
                                os << 0.01 * pdp->
>PowerDelayProfileValues.psValuesInDBm100[valIdx] << " ";
                        }
                    os << endl;
390                 }

                //
                // The CIR Peaks are now written to the console
                //
395     SViComList<SPeak>::SLinkedObject* peaks = p->pCir->ListOfPeaks.pFirst;
        while ( peaks )
        {
            os << "\t\tCIR Peak" << endl;
            os << "\t\t\tInband Power [dBm] = " << peaks->fInbandPowerInDBm <<
400 endl;

            os << "\t\t\tPeak Power [dBm] = " << peaks->fPeakPowerInDBm << endl;
            if ( peaks->sDopplerInHz != SPeak::sInvalidDopplerFrequencyInHz )
                os << "\t\t\tDoppler Frequency [Hz] = " << peaks->sDopplerInHz <<
405 endl;

            os << "\t\t\tDelay [sec] = " << peaks->fDelayInSec << endl;

            peaks = peaks->pNext;
        }
410     }

        //
        // Power Values with CINRS
        //
        const SViComList<SPowerValue>::SLinkedObject* pv = p->
415 >ListOfPowerValues.pFirst;
        while ( pv )
        {
            os << endl << "\tPower Value" << endl;

            if ( pv->pdwTimeFromStartOfBlockInNs )
                os << "\t\tTime from Start of Block [nsec] = " << * pv->
>pdwTimeFromStartOfBlockInNs << endl;

            os << "\t\tS-Sync Channel Power [dBm] = " << pv->fPowerInDBm << endl;
            //
            // The P-Sync power can be calculated if the S-Sync to P-Sync ratio
            // is known.
            //
425     if ( p->pfSSyncToPSyncRatioInDB )

```



```

os << "\t\tP-Sync Channel Power [dBm] = "
  << (pv->fPowerInDBm + * p->pfSSyncToPSyncRatioInDB)
  << endl;

435   if ( pv->pfCinrPSyncInDB )
os << "\t\tAveraged CINR [dB] = " << pv->fCinrInDB << endl;
else
os << "\t\tS-Sync CINR [dB] = " << pv->fCinrInDB << endl;

440   if ( pv->pfCinrPSyncInDB )
os << "\t\tP-Sync CINR [dB] = " << * pv->pfCinrPSyncInDB << endl;
if ( pv->pfAmpBasedCinrPSyncInDB )
os << "\t\tP-Sync (Amplitude based) CINR [dB] = " << * pv-
445 >pfAmpBasedCinrPSyncInDB << endl;
if ( pv->pfCinrSSyncInDB )
os << "\t\tS-Sync CINR [dB] = " << * pv->pfCinrSSyncInDB << endl;
if ( pv->pfAmpBasedCinrSSyncInDB )
os << "\t\tS-Sync (Amplitude based) CINR [dB] = " << * pv-
450 >pfAmpBasedCinrSSyncInDB << endl;

    pv = pv->pNext;
  }
}
455 //
// Wideband RS-CINR values
//
SViComList<SWidebandRsCinrResult>::SLinkedObject* widebandRsCinr =
res.ListOfWidebandRsCinrResults.pFirst;
460 while ( widebandRsCinr )
{
os << "\tWideband RS-CINR" << endl;
os << "\t\tScanner ID = " << widebandRsCinr->dwScannerBtsIdent << endl;
os << "\t\tbTransmitAntennaPort = " << (int)widebandRsCinr-
465 >bTransmitAntennaPort << endl;
os << "\t\t dwFrontEndSelectionMask = " << (int)widebandRsCinr-
>dwFrontEndSelectionMask << endl;
os << "\t\t Number of RB used = " << widebandRsCinr->wRBNumberOfBts <<
endl;
470 os << "\t\t Number of RE used = " << widebandRsCinr->dwCountOfUsedREs <<
endl;
os << "\t\tRSRP [dBm] = " << 0.01 * widebandRsCinr->sRSRPInDBm100 << endl;
os << "\t\tRSRQ [dB] = " << 0.01 * widebandRsCinr->sRSRQInDB100 << endl;
os << "\t\tRSRP [dBm] (noise clipping) = " << 0.01 * widebandRsCinr-
475 >sNoiseClippedRSRPInDBm100 << endl;
os << "\t\tRSRQ [dB] (noise clipping) = " << 0.01 * widebandRsCinr-
>sNoiseClippedRSRQInDB100 << endl;
if( widebandRsCinr->pWidebandRsCinrValues )
{
480 os << "\t\tRS-CINR Values" << endl;

for( DWORD dwRsCinrIndx = 0; dwRsCinrIndx < widebandRsCinr-
>dwCountOfSubbands; dwRsCinrIndx++ )
{
485 os << "\t\t[" << dwRsCinrIndx << "]: RS-CINR [dB] = " << 0.01 *
widebandRsCinr->pWidebandRsCinrValues[dwRsCinrIndx].sRsCinrInDB100 << endl;
// TODO: check for S_LTE_RS_CINR_VALUE::sInvalidValue
os << "\t\t\t68% confidence interval [dB]: ";
os << "To lower ratio: " << 0.1 * widebandRsCinr-
490 >pWidebandRsCinrValues[dwRsCinrIndx].ConfidenceInterval68PercentOfsRsCinrInDB100
.bDeviationToLowerRatioInDB10 << " ";
os << "To higher ratio: " << 0.1 * widebandRsCinr-
>pWidebandRsCinrValues[dwRsCinrIndx].ConfidenceInterval68PercentOfsRsCinrInDB100
.bDeviationToHigherRatioInDB10 << endl;
495 os << "\t\t\t95% confidence interval [dB]: ";
os << "To lower ratio: " << 0.1 * widebandRsCinr-
>pWidebandRsCinrValues[dwRsCinrIndx].ConfidenceInterval95PercentOfsRsCinrInDB100
.bDeviationToLowerRatioInDB10 << " ";
500 >pWidebandRsCinrValues[dwRsCinrIndx].ConfidenceInterval95PercentOfsRsCinrInDB100
.bDeviationToHigherRatioInDB10 << endl;

```

```

    }
}
505     widebandRsCinr = widebandRsCinr->pNext;
    }
    //
    // RSSI and spectrum values
    //
510     SViComList<SRssiAndSpectrumResult>::SLinkedObject* rssiAndSpectrum =
res.ListOfRssiAndSpectrumResults.pFirst;
    while ( rssiAndSpectrum )
    {
515         if( rssiAndSpectrum->psRssiInDBm100 )
        {
            os << "\tRSSI and Spectrum Result" << endl;
            os << "\t\tdwFrontEndSelectionMask = " << rssiAndSpectrum-
>dwFrontEndSelectionMask << endl;
            os << "\t\tRSSI Bandwidth [Hz] = " << rssiAndSpectrum-
520 >fUsedRssiBandwidthInHz << endl;
            os << "\t\tRSSI [dBm] = " << 0.01 * ( *rssiAndSpectrum->psRssiInDBm100
) << endl;
        }
        if( rssiAndSpectrum->pSpectrumResult )
525         {
            os << "\t\tSpectrum Frequency Spacing [Hz] = " << rssiAndSpectrum-
>fSpectrumFreqDistanceInHz << endl;
            os << "\t\tNumber of FFTs for spectrum[Hz] = " << rssiAndSpectrum-
>wCountOfFftsForSpectrum << endl;
530            os << "\t\t6-dB RBW [Hz] = " << rssiAndSpectrum-
>f6dBResolutionBandwidthInHz << endl;
            os << "\t\tPower values" << endl;
            for( DWORD dwSpectrumIndx = 0; dwSpectrumIndx < rssiAndSpectrum-
>pSpectrumResult->dwCountOfSpectrumValues; dwSpectrumIndx++ )
535                os << "\t\t[" << dwSpectrumIndx << "]: RMS [dBm] = " << 0.01 *
rssiAndSpectrum->pSpectrumResult-
>psRmsSpectralPowerValueInDBm100[dwSpectrumIndx] << endl;
        }
540         rssiAndSpectrum = rssiAndSpectrum->pNext;
    }
    //
    // MIMO results
    // Not supported yet!
545     //
    SViComList<SMimoResult>::SLinkedObject* mimo = res.ListOfMimoResults.pFirst;
    while ( mimo )
    {
550         os << "\tMIMO Result:" << endl;
            os << "\t\tScanner ID = " << mimo->dwScannerBtsIdent << endl;

            mimo = mimo->pNext;
        }
        //
555         // BCH demodulation results
        //
        if ( NULL != res.pPduResult )
        {
560             const SPduResult* pPDU = res.pPduResult;
            os << "PDU Result" << endl;
            os << "\tndwBtsId = " << pPDU->dwBtsId << endl;
            os << "\tdwFirstBtsId = " << pPDU->dwFirstBtsId << endl;
            os << "\tdwStartTimeInMs = " << pPDU->dwStartTimeInMs << endl;
            os << "\tdwStopTimeInMs = " << pPDU->dwStopTimeInMs << endl;
565             os << "\tePDU = " << pPDU->ePDU << endl;
            os << "\tdwBitCount = " << pPDU->dwBitCount << endl;
            os << "\twPhysicalCellId = " << pPDU->wPhysicalCellId << endl;
            // We don't display the bit stream pPDU->pbBitStream
        }
570         return os;
    }
}

```

```

575  /**
Worker function. The structure is pretty easy. In contrast to other sample
applications, the methods used here throw CViComError exception objects in
case that a problem arises, and these exceptions will be handled in the calling
method.
*/
580  void doLteMeasurement( CViComLteInterface& rInterface, const SSettings&
measConfig )
{
    DWORD const dwTimeout_ms = 15000;

    // Setting up the measurement
585    rInterface.GetBasicInterface().SetResultBufferDepth (
measConfig.ResultBufferDepth );
    rInterface.SetFrequencyTable (
measConfig.ChannelSettings );
    rInterface.SetBchDemodulationSettings (
590    measConfig.BchDemodulationSettings );

    // Starting the measurement
rInterface.GetBasicInterface().StartMeasurement();

595    // Getting measurement results
const SMeasResult* pRes = NULL;
unsigned int resultCount = 0;
while ( pRes = rInterface.GetResult(dwTimeout_ms) )
600    {
        if ( !_kbhit() )
            break;
        cout << string(30, '=') << " SMeasResult " << ++resultCount << endl;
        cout << *pRes << endl;

605        if ( NULL != pRes->pPduResult )
        {
            // Decode PDU and display the text
            CViComLteInterfaceData::SL3DecoderRequestData sRequest;
            sRequest.m_dwPDU = pRes->pPduResult->ePDU;
610            sRequest.m_dwBitCount = pRes->pPduResult->dwBitCount;
            DWORD dwByteCount = (sRequest.m_dwBitCount+7)/8;
            sRequest.m_pbBitStream = new BYTE[dwByteCount];
            memcpy( sRequest.m_pbBitStream, pRes->pPduResult->pbBitStream,
dwByteCount );
615            CViComLteInterfaceData::SL3DecoderResultData* pL3data =
rInterface.RetrieveTextForPDU( sRequest );
            if ( NULL != pL3data )
            {
                DWORD dwLen = pL3data->m_dwStringLength;
620                cout << "PDU Text >" << pL3data->m_pcStringPDU << endl;
            }
            delete [] sRequest.m_pbBitStream;
            sRequest.m_pbBitStream = NULL;

625        }

        // Stopping the measurement
rInterface.GetBasicInterface().StopMeasurement();
630    }
rInterface.GetBasicInterface().HasMeasurementStopped(10 * 1000);

    /**
Program entry point
*/
635    int _tmain(int argc, _TCHAR* argv[])
    {
        CViComLoader<CViComLteInterface, CViComLoader_TSMW> loader;
        try
        {
640            //
            // TSMW devices are connected using the IP address at which they can be
            // reached.
            //

```

```

645     loader.ConnectTsmw ("192.168.0.2");
        //
        // Assign the resources
        //
650     CViComBasicInterface&  rIfc = loader->GetBasicInterface();
        // Use frontend 1 with up to 50% load for narrowband scanning
        ResourceHandle        resourceNarrowbandScanner =
rIfc.GetTsmwResource(CViComBasicInterfaceData::TSMW_RF_FRONTEND_1, 50 );

        // Use frontend 1 with up to 50% load for wideband scanning
655     ResourceHandle        resourceWidebandScanner =
rIfc.GetTsmwResource(CViComBasicInterfaceData::TSMW_RF_FRONTEND_1, 50 );

        // Use frontend 2 with up to 100% load for demodulation
660     ResourceHandle        resourceDemodulator =
rIfc.GetTsmwResource(CViComBasicInterfaceData::TSMW_RF_FRONTEND_2, 100 );

        doLteMeasurement ( *loader, getLteMeasConfig( resourceNarrowbandScanner,
resourceWidebandScanner, resourceDemodulator ) );

665     //
        // Disconnecting the TSMW is done using the method shown below, not
        Release() known from the TSMx related CViComLoader classes
        //
        loader.DisconnectTsmw();
670     }
        catch (const CViComError& err)
        {
            cerr << "CViComError (" << err.GetErrorCode() << ") \" \" <<
err.GetErrorString() << '\n' << endl;
675     }
        return 0;
    }

```

The most interesting parts in the sample code are the initialization sequence (lines 462-467) of the TSMW, the configuration function `getLteMeasConfig()` (lines 35-191) and the result output in the overloaded operator `<<` (lines 200-452).

Initialization of the TSMW is here done with the default IP address of the TSMW. If this address has been changed, it must be adapted to the new setting to make the sample application run properly.

The measurement configuration is performed in the `getLteMeasConfig()` function. In this very simplistic scenario, it is configured to perform scanner measurement on a single frequency 806 MHz (line 39/76), and the demodulation of MIB and SIB1 is enabled (lines 159-187). The S-Sync/P-Sync ratio is not known, so a certain range is specified upfront (lines 128-130). Alternatively, the commented code fragment below can be utilized to set fixed ratios that shall be checked (lines 140-143).

The scanner measurement is setup to run on front-end 1, and the demodulation is also performed on the second front-end (see lines 524 resp. 530).

From all the other ViCom sample applications the main measurement loop (lines 474-499) is probably known. It simply process all results as long as there are any found within 10 seconds or unless a key is pressed. If a demodulation result is received, the byte buffer is decoded into a human-readable representation.

Result output is performed using the stream operator `<<`, which is overloaded and will print all valid values from the `SMeasResult` structure to stdout.



In comparison to the other sample applications, which introduce some exception handling by themselves, the ViCom LTE interface is the first to directly support exceptions in the interface. That is the reason why no specific `CViComError` instance is used throughout the code. All the Error handling is done in the central catch block (lines 539-542).

9 WiMAX Measurements

WiMAX stands for Worldwide Interoperability for Microwave Access. Based on the IEEE 802.16 standard, WiMAX provides ubiquitous broadband communication enabling cost-effective access to multiple kinds of networks regardless of whether they are fixed or mobile and allowing high throughput rates by robust implementation.

The downlink and uplink frame structures of WiMAX are shown below:

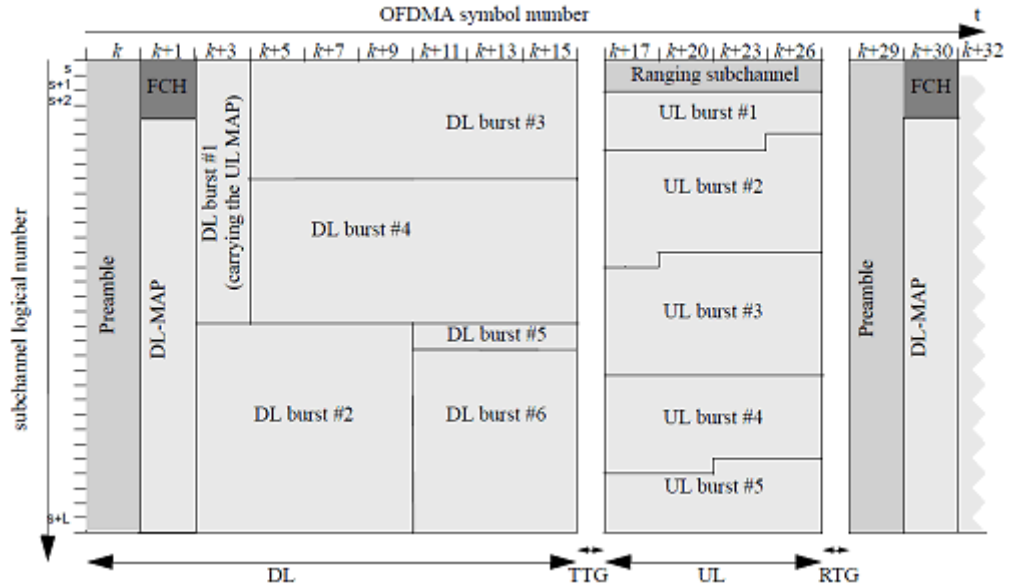


Figure 40 – WiMAX Downlink & Uplink Frame Structures

This ViCom interface utilizes TSMW to measure downlink WiMAX signals modulated using OFDMA (Orthogonal Frequency Division Multiple Access) in TDD (Time Division Duplex) mode.

In the following sections, the programming API (Application Program Interface) for TSMW and the WiMAX specific measurement details are described. Firstly, a short introduction into which and how WiMAX measurements are made in the ViCom is given. After that the sample console application coming with the ViCom delivery for WiMAX is explained.

9.1 Measuring WiMAX

The WiMAX ViCom module offers a simple and straight forward interface to provide a fast synchronization when a-priori knowledge about the signal is available. Each measure WiMAX signal result is stored in some easy-to-understand structures for the client application to access.

9.1.1 Configuration

The first step to do WiMAX measurements via the WiMAX ViCom module is to config its settings. Once the WiMAX ViCom module is properly configured, the client application can then make subsequent API calls to retrieve the measurement results from the TSMW scanner. These configurations are stored in the `SSettings` structure, which contains the following:

- Result Buffer Depth
- Channel Settings
- Demodulation Settings

9.1.1.1 Result Buffer Depth

After the `StartMeasrement()` function is called, whenever there is some result sent from TSMW, the WiMAX ViCom module stores it into some internal FIFO (First In, First Out) buffer. Then only when the `GetResult()` function is called, the “front” (earliest) result is pop out from this buffer. The size of this buffer (which must be between 1 to 1024 results) is configurable in the `SResultBufferDepth` structure within `SSettings`.

9.1.1.2 Channel Settings

The Channel Settings (`SChannelSettings`) mainly consists of a list of (up to 32) center frequencies and the corresponding bandwidths to be scanned. Another information it holds is the TSMW frontends. The TSMW contains two frontends that can be used to perform measurements on different channels in parallel. The measurement can be run on either one. The `resourceHandleForScanner` and `resourceHandleForDemodulator` variable hold the handles to the front-end used for scan and demodulation correspondingly. You can refer to the Resource Acquisition section for more info.

9.1.1.3 Demodulation Settings

The Demodulation Settings (`SBchDemodulationSettings`) specifies CINR threshold, the load percentage on TSMW to be used for demodulation (maximum value is 50) and the PDU Request Settings to start the measurement.

PDU Request Settings

The PDU Request Settings (`S_PDU_Requests`) specifies the following:

- On which channel to request PDU decode result (`dwChannelIndex`),
- The message type (`eMsgType`, should always be `MAC_MANAGEMENT_MESSAGE`) to be decoded.
- The type of PDU to be decoded (`ePDU`).
- PDU demodulation mode (`eDemodulationMode`) and its related parameters.

9.1.2 Measurement Result

Once the measurement has been set up, the measurement results are returned when requested using `GetResult()`. The WiMAX measurement results (`SMeasResult`) can be classified into two:

- WiMAX Scanner Results
- WiMAX Demodulation Results

9.1.2.1 WiMAX Scanner Results

WiMAX Scanner results contain information about the Channel Quality (RSSI and CINR) and Serving Cell (Preamble Index, Segment and ID Cell).

Channel Quality

- RSSI
The received signal strength indicator is the received wide band power in dBm, including thermal noise and noise generated in the receiver.
- CINR
The ratio of carrier/interference plus noise in dB.

RSSI and CINR are calculated from the power parameters of the Power Delay Profile inside the Channel Impulse Responses. For more details on CIR, please refer to the Channel Impulse Responses section in the LTE chapter.

Serving Cell

- Preamble Index
Downlink Preamble Index of the Base Station. It varies from 0 to 113 according to standard IEEE 802.16.
- Segment
A cell site can be divided up into three different segments. Thus segment number can be 0, 1 or 2.

ID Cell

ID of the cell; ranges from 0 to 31.

Downlink Frame Prefix

The downlink (DL) frame in WiMAX starts with preamble as its first symbol. In the DL symbols followed by preamble, the initial four subchannels are allocated for the Frame Control Header (FCH). The Downlink Frame Prefix (DLFP) is a data structure that contains information regarding the current frame and is mapped to the FCH. More specifically the DLFP contains the following information:

- Bitmap of Used Subchannels
- Length of Downlink Map (DL-MAP) that immediately follows the DLFP
- DL-MAP Repetition Coding (0, 2, 4 or 6)
- DL-MAP Encoding (such as CC, BTC, CTC, ZT CC, CC with interleaver and LDPC)

The FCH is always coded with the QPSK rate 1/2 mode with four repetitions to ensure maximum robustness and reliable performance, even at the cell edge.

Downlink Map

The Downlink Map (DL-MAP) message (Management Message Type = 2) defines the access to the DL information. It contains the following information:

- DCD Count
- No. OFDMA Symbols

DL-MAP also contains some IEs, of which we are interested in the Downlink Burst Profile (whose DIUC is between 0 and 12, both inclusive). It contains the following information:

- OFDMA Symbol Offset
- Subchannel Offset
- Boosting
- No. OFDMA Symbols
- No. Subchannels
- Repetition Coding

Uplink Map

The Uplink Map (UL-MAP) message (Management Message Type = 3) allocates access to the UL channel. It contains the following information:

- UCD Count
- Allocation Start Time
- No. OFDMA Symbols

UL-MAP also contains some IEs, of which we are interested in the CDMA BR, CDMA ranging (whose UIUC is 12). It contains the following information:

- OFDMA Symbol Offset
- Subchannel Offset
- No. OFDMA Symbols
- No. Subchannels
- Ranging Method
- Dedicated Ranging Indicator

Compressed Map

In addition to the standard DL-MAP and UL-MAP formats described above, there are also Compressed Maps. The presence of the Compressed DL-MAP format is indicated by the contents of the most significant three bits of the first data byte. When this combination of three bits is set to 110 (an invalid combination for a standard header in the downlink), the compressed DL-MAP format is present. A compressed UL-MAP shall only appear after a compressed DL-MAP. The presence of a compressed UL-MAP is indicated by a bit in the compressed DLMAP data structure.

Compressed DL-Map

The Compressed DL-MAP presents the same information as the standard format with one exception. In place of the DL-MAP's 48-bit Base Station ID parameter, the compressed format provides a subset of the full value, i.e. Operator ID and Sector ID. The Operator ID holds the 8 LSBs of the 24 MSBs of the 48-bit Base Station ID parameter, and the sector ID holds the 8 LSBs of the 48-bit Base Station ID parameter. When the compressed format is used, the full 48-bit Base Station ID parameter shall be published in the DCD.

Compressed UL-Map

The Compressed Uplink Map (UL-MAP) format may only appear after a compressed DL-MAP message to which it shall be appended. The message presents the same information as the standard format with the exception that the generic MAC header is omitted.

DL Channel Descriptor & UL Channel Descriptor

A DL Channel Descriptor (DCD) is transmitted by the BS at a periodic interval to define the characteristics of a DL physical channel. A UL Channel Descriptor (UCD) is transmitted by the BS at a periodic interval to define the characteristics of a UL physical channel. They contain information described below.

DCD

- Configuration Change Count

UCD

- Configuration Change Count
- Ranging Backoff Start
- Ranging Backoff End
- Request Backoff Start
- Request Backoff End

Both DCD and UCD may contain optional TLV fields, some of them are described as follows

DCD

- Transmit/Receive Transition Gap (TTG)
- Receive/Rransmit Transition Gap (RTG)
- BS EIRP in dBm
- EIRxPIR,max
- Base Station ID in case of compressed DLMAP.
- MAC Version

9.1.2.2 WiMAX Demodulation Results

The WiMAX demodulation results are the decoded contents of the PDU of MAC Management message. To get demodulation results, the client application first needs to check to make sure that `pPduResult` field of `SMeasResult` is no NULL (i.e. it contains valid PDU data). And then it uses this PDU data to fill the `SL3DecoderRequestData` structure and uses that structure to call the `RetrieveTextForPDU` function to retrieve the demodulation results. The results (`SL3DecoderResultData`) will be filled in with variables containing decoded contents and a descriptive string of all these contents (`pcStringPDU`) for the PDU requested, which can be DL-MAP, UL-MAP, DCD, UCD and so on.

9.1.3 Error Handling

The WiMAX ViCom interface supports C++ exceptions natively. For every method in the interface (and also in the basic interface) taking a `CViComError` reference to be filled with potential error messages, there is a counterpart method that will throw a `CViComError` object in case that a problem arises. All the remaining behavior is consistent between both methods, so you can choose which version fits your needs best.[^]

9.2 GUI Sample Application

The sample application is a simple implementation of the ViCom interface, to control one WiMax scanner. It may be used

- To test the R&S TSMW scanner
- To understand the ViCom programming interface

The application and its user interface was principally designed for testing the ViCom interface functions so it is possible to call interface functions at any time from the GUI, and it is possible to set values out of range to check the responses and the behaviour of the PN-Scanner in different situations. This means the code is quite straightforward and provides a good example of how to program a measurement application.

Ensure that ROMES demo is not currently running, and double click on the executable file `SampleForViComWiMaxRelease.exe`.

There is only one screen in the test application. This is shown below.

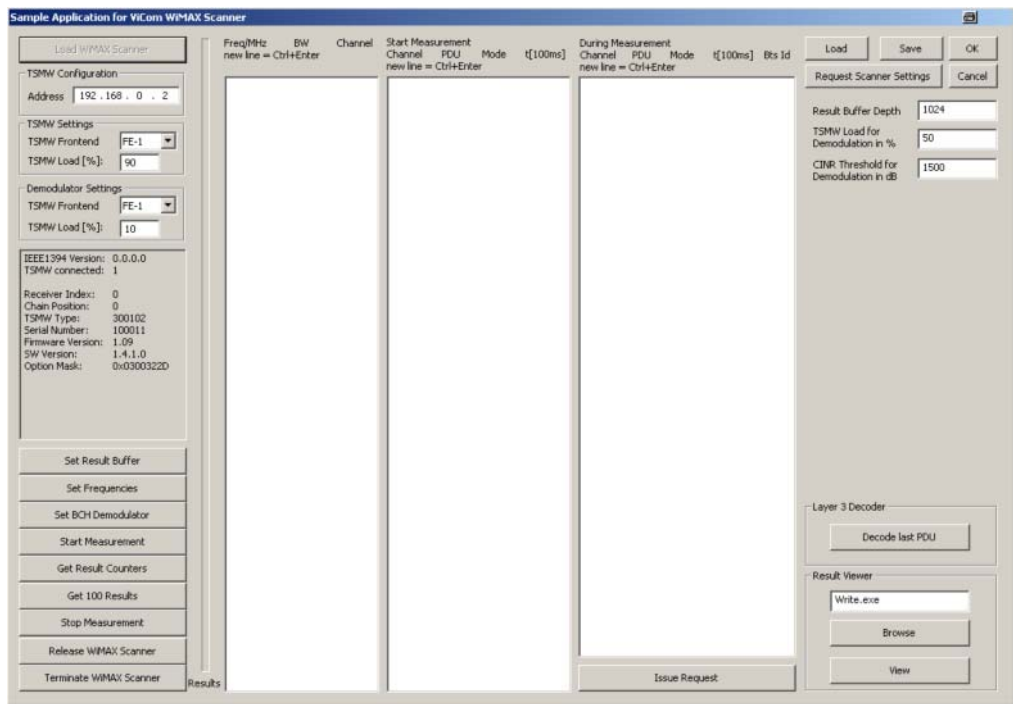


Figure 41 – the WiMax sample

9.2.1 Setup connection

The first action after starting the sample application, should be to enter the IP address of the TSMW and click the “Load WiMAY Scanner” button, highlighted below.

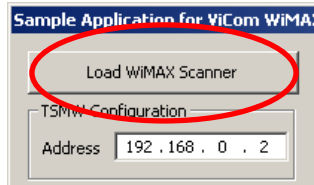


Figure 42 – dialog to load the scanner

If the WiMAX Scanner is connected and configured with the default IP address then clicking “Load WiMAX Scanner” will cause the sample application to load a ViCom WiMAX interface (See section 3 “Programming with the ViCom Interface” for more explanation about loading the interface). When the scanner is loaded successfully, information similar to that shown in the figure below, will appear in the text box below the Demodulator settings. Note that the button is now greyed out to indicate that a device is now loaded.

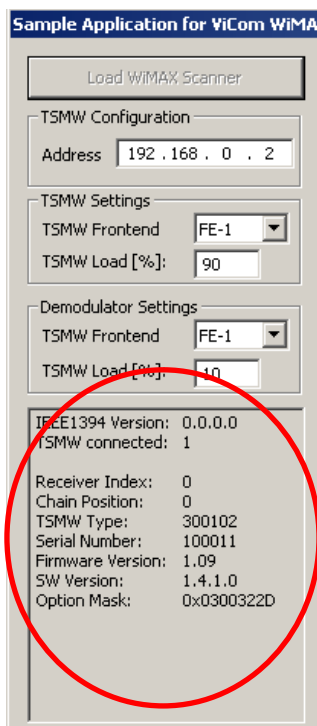


Figure 43 – scanner information

Above this information box are two groups of settings:

- TSMW settings
- Demodulator settings

The TSMW settings tell the scanner what the maximum load for measurements is for each Front End (FE). The TSMW has 2 front ends and each can be configured using the pull-down list. In this case it can be seen that the maximum load is set to 90%.

The Demodulator settings tell the scanner what the maximum load for BCH demodulation is for each front end. In this case it can be seen that the maximum load is set to 10%.

9.2.2 Measurement controls

The interface functions can now be accessed by clicking on the buttons shown below.

The parameters to be used by the scanner (e.g. Frequency and bandwidth) can be entered and edited in the columns in the center of the GUI.

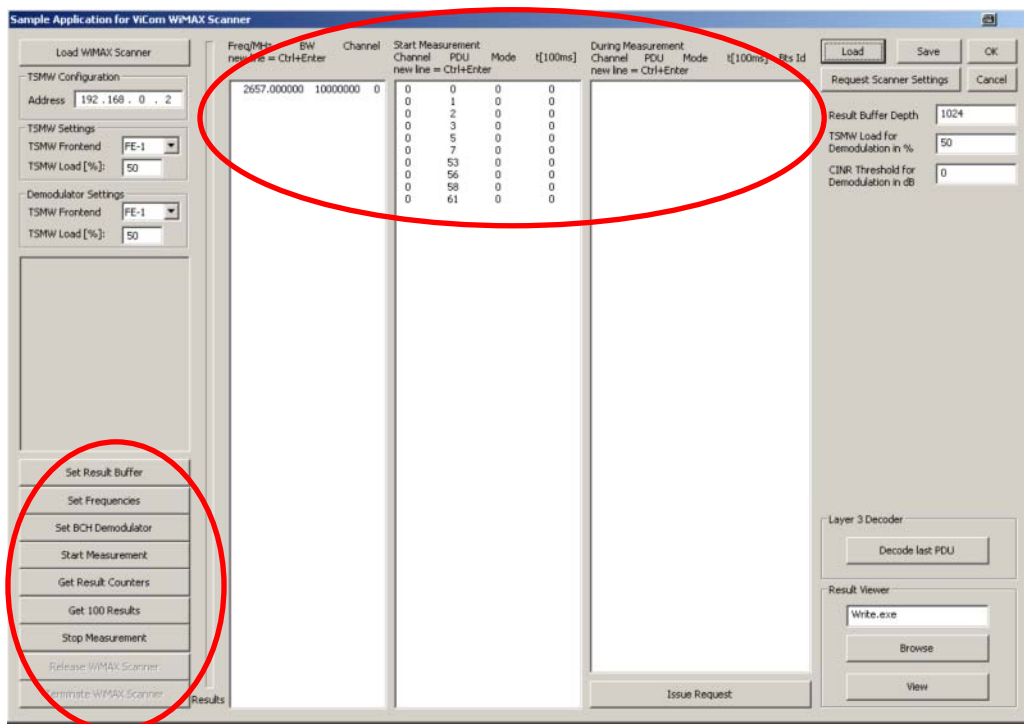


Figure 44 – control buttons and parameter fields

The “Set..” buttons correspond to interface functions that set parameters in the scanner. An initial set of default values appear in the settings fields (the white columns in the figure above). These can be edited in the fields, and sent to the scanner by clicking the corresponding “Set” button.

Note that the scanner has its own set of default values for all parameters except the frequencies. The scanner will use its own default values, unless they have been overridden by interface function calls using the dialogue buttons. Because there is no default value for the frequency in the scanner, at least one frequency must be set using the “Set Frequencies” dialogue button, before any measurements can be made.

The Scanner settings remain in the scanner until new interface functions are called. The only exception is the settings for the BCH demodulator. These settings are channel specific and reset to default values when the frequency settings are changed.

The other buttons in the figure above correspond to interface functions that deal with measurements, and with unloading the scanner. “Start Measurement” tells the system to start measuring. “Get Result Counters” causes the current number of measurement results in the ViCom interface buffer to be displayed, as well as the number of measurements that have been deleted if the buffer has overflowed.

“Get next 100” causes the application to get the next 100 measurement results from the interface buffer, The results will be written to the file ViComMeasurements, in the sub-directory \Application\LogFiles.

After making a measurement, the TSMW can be unloaded using the “Release WiMAX Scanner” button.

If problems are encountered during the release procedure, the “Terminate WiMAX Scanner” button can be used. The ViCom API then forces a deletion of the `ViComWiMAXInterface` object.

The figure below shows the columns in the central area of the GUI. The frequency to be scanned can be entered in the left column along with the local channel identifier.



Frequencies default

Please note, that although a default value appears in the GUI, this is not entered in the scanner until the “Set Frequencies” button has been clicked.

Freq/MHz		BW	Channel	Start Measurement				During Measurement				
new line = Ctrl+Enter				Channel	PDU	Mode	t[100ms]	Channel	PDU	Mode	t[100ms]	Bts Id
				new line = Ctrl+Enter				new line = Ctrl+Enter				
2657.000000	10000000	0	0	0	0	0	0					
			0	1	0	0	0					
			0	2	0	0	0					
			0	3	0	0	0					
			0	5	0	0	0					
			0	7	0	0	0					
			0	53	0	0	0					
			0	56	0	0	0					
			0	58	0	0	0					
			0	61	0	0	0					

Figure 45 – columns in the GUI

The center column lists the PDU’s which correspond to the management messages which the scanner should demodulate. Each PDU value must have a corresponding channel identifier which refers to one of the frequencies entered in the left hand box.

9.2.3 View Results

Stored results can be browsed and viewed using the dialogue in the View Results box. Enter the file name of your preferred text editor in the text field - for example “notepad.exe” (the “Browse” button can be used to find the program if need be). Click the “View” button, and the sample application will open the result file using the preferred text editor.

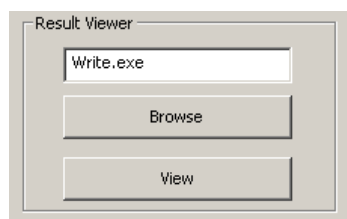
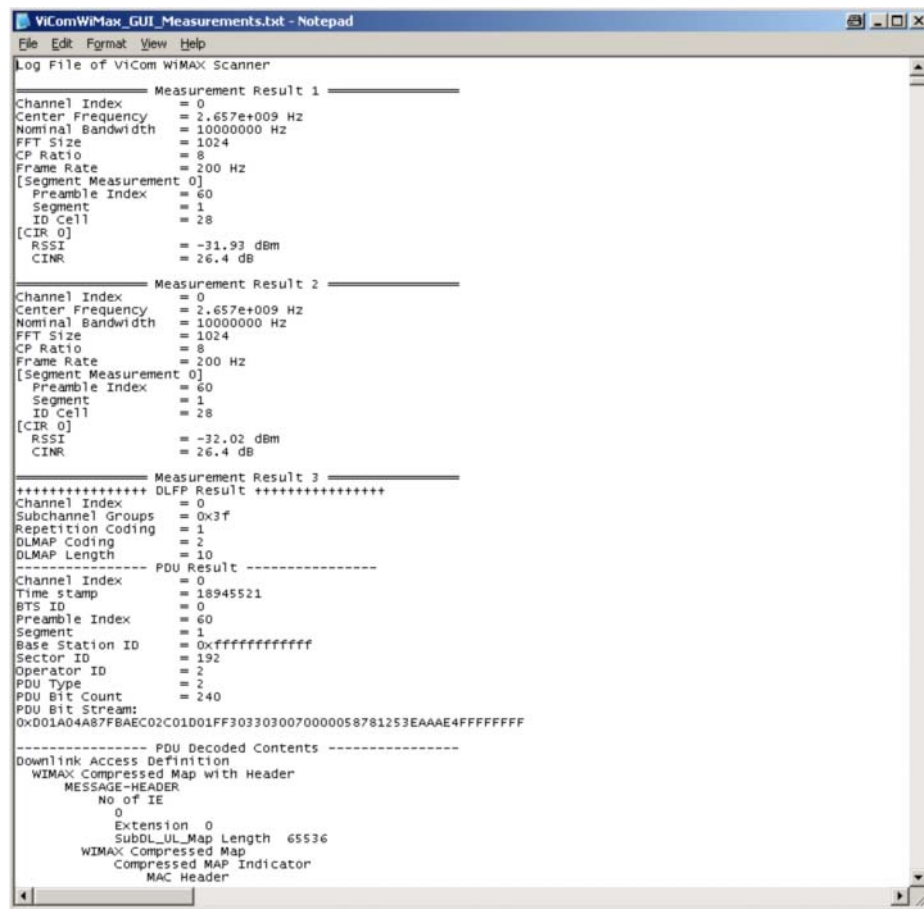


Figure 46 – the result viewer

A sample of a R&S TSMW measurement file is shown below. For an explanation of the fields listed, please see section 5, ViCom data fields, and the SMeasResult structure.



```

ViComWiMax_GUI_Measurements.txt - Notepad
File Edit Format View Help
Log File of ViCom WiMAX Scanner
----- Measurement Result 1 -----
Channel Index          = 0
Center Frequency       = 2.657e+009 Hz
Nominal Bandwidth     = 10000000 Hz
FFT Size              = 1024
CP Ratio              = 8
Frame Rate            = 200 Hz
[Segment Measurement 0]
  Preamble Index       = 60
  Segment              = 1
  ID Cell              = 28
[CIR 0]
  RSSI                 = -31.93 dBm
  CINR                 = 26.4 dB
----- Measurement Result 2 -----
Channel Index          = 0
Center Frequency       = 2.657e+009 Hz
Nominal Bandwidth     = 10000000 Hz
FFT Size              = 1024
CP Ratio              = 8
Frame Rate            = 200 Hz
[Segment Measurement 0]
  Preamble Index       = 60
  Segment              = 1
  ID Cell              = 28
[CIR 0]
  RSSI                 = -32.02 dBm
  CINR                 = 26.4 dB
----- Measurement Result 3 -----
+++++ DLFP Result +++++
Channel Index          = 0
Subchannel Groups     = 0x3f
Repetition Coding     = 1
DLMAP Coding          = 2
DLMAP Length         = 10
----- PDU Result -----
Channel Index          = 0
Time stamp            = 18945521
BTS ID                = 0
Preamble Index       = 60
Segment              = 1
Base Station ID       = 0xffffffff
Sector ID             = 192
Operator ID           = 2
PDU Type              = 2
PDU Bit Count         = 240
PDU Bit Stream:
0x001A04A87FBAEC02C01D01FF3033030070000058781253EAAAE4FFFFFFFF
----- PDU Decoded Contents -----
Downlink Access Definition
WIMAX Compressed Map with Header
MESSAGE-HEADER
  No. of IE
  0
  Extension 0
  subDL_UL_Map Length 65536
WIMAX Compressed Map
  Compressed MAP Indicator
  MAC Header

```

Figure 47 – example of a measurement file

9.2.4 Update GUI with current scanner settings

The “Request Scanner Settings” button is shown below and when clicked displays the values of all settings which are in use in the scanner at that moment. This button can be clicked after starting a measurement, and will display the settings in the appropriate dialogue fields. The settings can also be stored in an R&S TSMW settings file (*.tsm), and later reloaded, using the “Save” and “Load” buttons respectively.

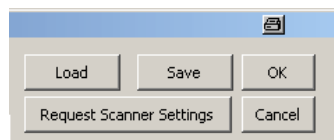


Figure 48 – scanner settings

9.2.5 Miscellaneous

The “Cancel” and “OK” buttons both terminate the application. “OK” stores the settings in a .bin”file in the application directory before terminating. If available, this file is used to initialise the application, (not the scanner!), after reloading.

To see the code of the test application

- go to the application folder: C:\RUS\ViCom_14.30\SampleForViComWiMax_cmd
- open the project file: SampleForViComWiMax_cmd.vcproj

9.2.6 Demodulation

In the sample application it is possible to experiment with the usage of the WiMAX ViCom API to demodulate the BCH channel and decode the demodulated PDUs.

In this section, a complete walk-through is shown to help you getting a first demodulation result. So if you follow the steps shown below, you should get the BCH demodulation up and running very fast and also gain an impression of how to use the API.

The important parts of the demo application concerning the BCH demodulator are the center and right-handed columns. If dynamic requests are to be issued during measurement, the Issue Request button can be used.

Once the application has been started, the following operation sequence should be performed to do a first sample measurement:

- Load the TSMW. This can be done by clicking on “Load Scanner” button. If this command was successful, the list box is filled with some details of the device.
- Click the buttons “Set Result Buffer” to set the “Result Buffer Depth” in the TSMW device.
- Define the frequency table and set the table using the “Set Frequencies” button. In the table, space-delimited pairs of a frequency and a channel index can be defined. For a simpler editing, it is also possible to add the frequencies only, one per line (a new line is started with Ctrl+Enter). The channel indices will be added automatically when “Set Frequencies” is pressed.
- Specify the demodulation settings. Each line in the central list box contains demodulation settings for one channel/PDU pair. The numbers in the columns have the meanings defined below:
 - The first number is the channel number that can be seen in the frequency list box. It is simply used as a label to represent a particular frequency – this avoids having to work with the actual frequency numbers (which is inefficient and error prone).
 - The second number is the internal PDU name.
 - The third number is the demodulation mode.
 - The fourth and final number is a timeout, if the demodulation mode is set to 2 (which means repetition mode). The value here is entered in 100 milliseconds. For example, a value of 10 defines $10 * 100 \text{ ms} = 1 \text{ second}$ as time-out.
- Start the measurement. If you now click on “Get Result Counters” button regularly, you will find that new data is collected after the measurement has started.
- Fetch the results with “Get 100 Results. Display the output file (using “View” button). The file is stored in the ViCom root dir in the file “Application\Logfiles\ViComMeasurements.txt”. If the demodulation could be done within the 100 results there should be some decoded Layer-3 messages in the text file. You can search for “L3” in the file to find demodulation results faster.

Issuing a Request during Measurement

It is possible to explicitly request a demodulation of a certain cell during measurement.

To do this you need to:

- In the “During Measurement” column, enter the same four parameters described in the previous section, along with the Cell Id.
- Click on the button “Issue Request” to send the request to the demodulator.
- Click on “Get 100 Results” and check the result file as described above for the demodulated string that has been requested. Be sure to close any previously started instances of your file viewer (default: write.exe) if the editor does not support automatic file updates (write.exe does NOT support this feature).

When you finished working with the demo application, don't forget to stop the measurement and release the resources of the TSMW.

9.3 Command Line Sample Application

The WiMAX ViCom API comes with a simple command line application that demonstrates the usage of the API. It can be found in the ViCom installation folder in the directory called “SampleForViCom WiMAX_cmd”.

This command line sample application is structured similarly as the Code Examples in the LTE ViCom. It mainly starts a measurement, retrieves the measurement result with 10 seconds timeout and prints the signal details to the console, until a key is pressed. Here are the code samples followed by some explanatory notes.

NOTICE

Text wrapping & Line numbering

Please be aware that some of the lines in the code are too long to be displayed properly in a text document. These code lines are wrapped in this document and this affects the indentation and line numbering.

For example: lines 92 & 93 are a single line of code although the doc counts them as separate lines due to the text wrapping.

```
// SampleForVicomWiMax_cmd.cpp : Defines the entry point for the console
application.
//
5 #include "stdafx.h"
#include <conio.h>
#include <math.h>
#include <assert.h>
10 #include <typeinfo.h>
#include <algorithm>
#include <cctype>
#include <vector>
#include <iomanip>
15 #include <ViComWiMaxInterface.h>
#include <ViComLoader.h>
```

```

using namespace std;

20 // Error display macro
#define DUMP_VICOM_ERROR(ViComError) \
    cerr << __FILE__ << "(" << __LINE__ << ") : " \
    << __FUNCTION__ << "(): CViComError (" << err.GetErrorCode() << ") \"\" \
    << err.GetErrorString() << "\"\" << endl;
25
// typedef for less typings.
typedef CViComBasicInterfaceData::DeviceResourceHandle ResourceHandle;
typedef CViComBasicInterfaceData::SConnectedReceiverTable
SConnectedReceiverTable;
30 typedef CViComBasicInterfaceData::etTsmwFrontendId etTsmwFrontendId;
typedef CViComBasicInterface::SMessage SMessage;
typedef CViComWiMaxInterfaceData::SChannelSettings SChannelSettings;
typedef CViComWiMaxInterfaceData::SFrequencySetting SFrequencySetting;
typedef CViComWiMaxInterfaceData::SBchDemodulationSettings
35 SBchDemodulationSettings;
typedef CViComWiMaxInterfaceData::SResultBufferDepth SResultBufferDepth;
typedef CViComWiMaxInterfaceData::SSettings SSettings;
typedef CViComWiMaxInterfaceData::SMeasResult SMeasResult;
typedef SMeasResult::SSegmentMeas SSegmentMeas;
40 typedef SMeasResult::SCir SCir;
typedef SCir::SPowerDelayProfile SPowerDelayProfile;
typedef SCir::SPeak SPeak;
typedef SMeasResult::SDlfp SDlfp;
typedef SMeasResult::SPduResult SPduResult;
45 typedef CViComWiMaxInterfaceData::etPDU_DEMOD_MODE etPDU_DEMOD_MODE;
typedef CViComWiMaxInterfaceData::etWiMaxMessageType etWiMaxMessageType;
typedef CViComWiMaxInterfaceData::etMacManagementPDU etMacManagementPDU;
typedef CViComWiMaxInterfaceData::S_PDU_Requests S_PDU_Requests;
typedef S_PDU_Requests::S_WiMax_PDU_Request S_WiMax_PDU_Request;
50 typedef CViComWiMaxInterfaceData::SL3DecoderRequestData
SL3DecoderRequestData;
typedef CViComWiMaxInterfaceData::SL3DecoderResultData
SL3DecoderResultData;
typedef SL3DecoderResultData::SDlmap SDlmap;
55 typedef SDlmap::SDlmapIE SDlmapIE;
typedef SDlmapIE::SIEBurstProfile SIEBurstProfile;
typedef SL3DecoderResultData::SULmap SULmap;
typedef SULmap::SULmapIE SULmapIE;
typedef SULmapIE::SIERanging SIERanging;
60 typedef SL3DecoderResultData::SDcd SDcd;
typedef SL3DecoderResultData::SUcd SUcd;
typedef SL3DecoderResultData::STlv STlv;
typedef SL3DecoderResultData::SMobNbrAdv SMobNbrAdv;
typedef SMobNbrAdv::SNeighbor SNeighbor;
65 typedef STlv::SDownlinkBurstProfile
SDownlinkBurstProfile;
typedef STlv::SUplinkBurstProfile SUplinkBurstProfile;
typedef SL3DecoderResultData::SRngRsp SRngRsp;
typedef SL3DecoderResultData::SRegRsp SRegRsp;
70 typedef SL3DecoderResultData::SMobBshoReq SMobBshoReq;
typedef SL3DecoderResultData::SMobMshoReq SMobMshoReq;
typedef SL3DecoderResultData::SMobBshoRsp SMobBshoRsp;
typedef SL3DecoderResultData::SMobScnRep SMobScnRep;
typedef SL3DecoderResultData::SMobPagAdv SMobPagAdv;
75

// File scope globals
static string g_sAddr = "192.168.0.2";
static vector<double> g_dCenterFreqInHzVector;
80 static vector<DWORD> g_dwBandWidthVector;
static short g_sCINRThresholdInDB100 = 1500; // Default
set to 15.0 dB.
static vector<DWORD> g_dwChannelIndex; // Holds the channel
index for the corresponding PDU type.
85 static vector<etMacManagementPDU> g_ePduVector;
// Set the PDU type for TLV display as there is no way to pass this information
in overloading the << operator.

```

```

static etMacManagementPDU          g_ePduForTlvDisplay;
// Specify which front-end to be used for scanning and which front-end to be
90 used for demodulation.
// By default, front-end one is used for scanning and front-end two is used for
demodulation.
static string                      g_sFrontEnds = "12";

95 // Utility functions =====>
/**
  Get the string to display the frequency in a more user-friendly way.
  @param[in] dCenterFreqInHz  Frequency in Hz.
  @return A std::string containing the user-friendly representation of the
100 frequency.
  */
string GetUserFriendlyFreq(double dwFreqInHz)
{
105     ostringstream oss;

    if(dwFreqInHz < 1000) {
        oss << dwFreqInHz << " Hz";
    } else if(dwFreqInHz < 1000 * 1000) {
110     } else if(dwFreqInHz < 1000 * 1000 * 1000) {
        oss << dwFreqInHz / (1000 * 1000) << " MHz";
        // .0 to avoid warning C4307: '*' : integral constant overflow.
    } else if(dwFreqInHz < 1000.0 * 1000 * 1000 * 1000) {
115     } else { // too big? still use GHz as the unit.
        oss << dwFreqInHz / (1000 * 1000 * 1000) << " GHz";
    }

    return oss.str();
120 }

    ///! BadConversion exception defined to be used by StringToNumber(std::string
const& s)
class BadConversion : public std::runtime_error {
125 public:
    ///! Constructor
    BadConversion(std::string const& s)
        : std::runtime_error(s)
    { }
130 };

/**
  Convert a string to the given type NUM_TYPE.
  @param[in] s  A reference to the string to be converted.
135 @return      The converted data (of NUM_TYPE).
http://www.parashift.com/c++-faq-lite/misc-technical-issues.html#faq-39.2
  */
template<class NUM_TYPE>
inline NUM_TYPE StringToNumber(std::string const& s)
140 {
    std::istringstream i(s);
    NUM_TYPE x;
    if (!(i >> x)) {
145     throw BadConversion("StringToNumber<" +
std::string(typeid(NUM_TYPE).name()) + ">(\"" + s + "\")");
    }
    return x;
}

150 // <==== Utility functions

/**
  Parse the given argument string and set the corresponding option of
the program if the argument string is valid.
155 @param[in] arg  The given argument string.
  @return      true if the given argument string contains a valid option/switch.
                false if the given argument string contains no valid option/switch.
  */

```

```

bool ParseArg(char* arg) {
160   string s(arg);
      // ToUpper() for std::string
      // http://www.dreamincode.net/forums/topic/15095-convert-string-to-uppercase-
in-c/
      transform(s.begin(), s.end(), s.begin(), std::toupper);
165
      const string addrPrefix("-ADDR=");
      const string cfPrefix("-CF=");
      const string bwPrefix("-BW=");
      const string cinrPrefix("-CINR=");
170   const string pduPrefix("-PDU=");
      const string fePrefix("-FE=");

      if(s.substr(0, addrPrefix.length()) == addrPrefix) {
          g_sAddr = s.substr(addrPrefix.length());
175   } else if(s.substr(0, cfPrefix.length()) == cfPrefix) {
          string& sCf = s.substr(cfPrefix.length());
          try {
              g_dCenterFreqInHzVector.push_back(StringToNumber<double>(sCf));
          } catch (BadConversion&) {
180             cerr << "Invalid center frequency specified: " << sCf << endl
                  << "  The center frequency must be specified in HZ" <<endl;
          }
      } else if(s.substr(0, bwPrefix.length()) == bwPrefix) {
          string& sBwArg = s.substr(bwPrefix.length());
185   if(sBwArg == "3.5M") {
              g_dwBandWidthVector.push_back(SFrequencySetting::WIMAX_BW_3_5);
          } else if(sBwArg == "5M") {
              g_dwBandWidthVector.push_back(SFrequencySetting::WIMAX_BW_5);
          } else if(sBwArg == "7M") {
190             g_dwBandWidthVector.push_back(SFrequencySetting::WIMAX_BW_7);
          } else if(sBwArg == "8.75M") {
              g_dwBandWidthVector.push_back(SFrequencySetting::WIMAX_BW_8_75);
          } else if(sBwArg == "10M") {
195             g_dwBandWidthVector.push_back(SFrequencySetting::WIMAX_BW_10);
          } else {
              cerr << "Invalid bandwidth specified: " << sBwArg << endl
                  << "  Must be one of: 3.5M | 5M | 7M | 8.75M | 10M" <<endl;
          }
      } else if(s.substr(0, cinrPrefix.length()) == cinrPrefix) {
          string& sCinr = s.substr(cinrPrefix.length());
          try {
              g_sCINRThresholdInDB100 = (WORD)(StringToNumber<double>(sCinr) * 100);
          } catch (BadConversion&) {
200             cerr << "Invalid CINR threshold specified: " << sCinr << endl;
          }
      } else if(s.substr(0, pduPrefix.length()) == pduPrefix) {
          string& sPdu = s.substr(pduPrefix.length());
          try {
210             g_ePduVector.push_back((etMacManagementPDU)StringToNumber<int>(sPdu));
              // If no center frequency specified yet, set the corresponding channel
              // index to zero,
              // as later we will guarantee there is at least once frequency in
              // g_dCenterFreqInHzVector in GetArgOptions().
              g_dwChannelIndex.push_back(max(0, g_dCenterFreqInHzVector.size() - 1));
215   } catch (BadConversion&) {
              cerr << "Invalid ePDU specified: " << sPdu << endl
                  << "  The ePDU must be specified in integer" <<endl;
          }
      } else if(s.substr(0, fePrefix.length()) == fePrefix) {
          string& sFe = s.substr(fePrefix.length());
          if(sFe == "11" || sFe == "12" || sFe == "21" || sFe == "22")
          {
              g_sFrontEnds = sFe;
          } else {
220             cerr << "Invalid front-end usage specified: " << sFe << endl
                  << "  The front-end usage must be specified as one of the
following:" << endl
                  << "  11, 12, 21, 22." << endl
            }
225

```

```

        << " The first number specifies which front-end to be used for
230 scanning," << endl
        << " The second number specifies which front-end to be used for
demodulation." << endl;
    }
    } else {
235     cerr << "Invalid argument: " << arg << endl;
        return false;
    }

    return true;
240 }

/**
Set the options according to the program arguments.
@param[in] argc The argc in C/C++.
245 @param[in] argv The argv in C/C++.
@return true if the given argument string contains a valid option/switch.
false if the given argument string contains no valid option/switch.
*/
bool GetArgOptions(int argc, char* argv[])
250 {
    for(int i = 1; i < argc; i++) {
        if(!ParseArg(argv[i])) {
            return false;
255     }
    }

    // Set at least one default frequency.
    if(g_dCenterFreqInHzVector.empty()) {
260     g_dCenterFreqInHzVector.push_back(2600. * 1000 * 1000);
    }

    // Make sure the bandwidth list and the frequency list match each other.
    for(vector<double>::size_type i = g_dwBandWidthVector.size(); i <
g_dCenterFreqInHzVector.size(); i++) {
265     g_dwBandWidthVector.push_back(SFrequencySetting::WIMAX_BW_10); // Set 10M
for those frequencies with unspecified BW.
    }

    for(vector<DWORD>::size_type i = g_dCenterFreqInHzVector.size(); i <
270 g_dwBandWidthVector.size(); i++) {
        g_dwBandWidthVector.pop_back();
    }

    return true;
275 }

/**
Create the measurement configuration for the measurement scenario.
@param[in] resourceScanner Resource handler for the scanner.
280 @param[in] resourceDemodulator Resource handler for the demodulator.
@return A SSettings structure to config the measurement.
*/

SSettings getWiMaxMeasConfig (ResourceHandle resourceScanner, ResourceHandle
285 resourceDemodulator)
{
    SSettings settings = SSettings();
    settings.ResultBufferDepth.dwValue = settings.ResultBufferDepth.dwMax;

290     for (DWORD i = 0; i < g_dCenterFreqInHzVector.size(); i++)
    {
        settings.ChannelSettings.dwCount = i+1;
        SFrequencySetting& s =
settings.ChannelSettings.aTableOfFrequencySetting[i];
295
        //
        // Select the RF frontend used to measure this frequency.
        //
        s.resourceHandleForScanner = resourceScanner;

```

```

300     s.resourceHandleForDemodulator = resourceDemodulator;

        //
        // The frequency that shall be measured, in Hertz and as floating
        // point number
305     //
        s.dFreqRangeMinInHz = g_dCenterFreqInHzVector[i];
        s.dFreqRangeMaxInHz = g_dCenterFreqInHzVector[i];

310     s.dwNomBandWidthRangeMinInHz = g_dwBandWidthVector[i];
        s.dwNomBandWidthRangeMaxInHz = g_dwBandWidthVector[i];
    }

    DWORD dwRequests = g_ePduVector.size();
    SBchDemodulationSettings& rBCH =
315 settings.BchDemodulationSettings;
        rBCH.sCINRThresholdInDB100 =
            g_sCINRThresholdInDB100;
        rBCH.wLoadInPercent =
            rBCH.wOptLoadInPercent;
320 rBCH.sStartMeasurementRequests.dwCountOfPDURRequests = dwRequests;
        rBCH.sStartMeasurementRequests.pPDURRequest = new
            S_WiMax_PDU_Request[dwRequests];
        memset( rBCH.sStartMeasurementRequests.pPDURRequest, 0,
            rBCH.sStartMeasurementRequests.dwCountOfPDURRequests *
325 sizeof(S_WiMax_PDU_Request) );

        for(DWORD i = 0; i < dwRequests; i++)
        {
            S_WiMax_PDU_Request& rRequest =
330 rBCH.sStartMeasurementRequests.pPDURRequest[i];
                rRequest.dwChannelIndex = g_dwChannelIndex[i];
                rRequest.eMsgType =
                    CViComWiMaxInterfaceData::MAC_MANAGEMENT_MESSAGE;
                rRequest.ePDU = g_ePduVector[i];
335 rRequest.eDemodulationMode =
                    CViComWiMaxInterfaceData::PDU_DEMOD_ONCE;
                rRequest.dwRepetitionTimeOutInMs = 0;
                rRequest.dwBtsId = 0;
        }
340     return settings;
    }

    /**
345 Helper operator to write information onto console.
        You can extend this to add additional output when you are interested in more
        details on the selected signals.
        @param[in] os The ostream (normally, cout).
        @param[in] res The SMeasresult struct to be dumped.
350 @return A reference to os.
    */
    ostream& operator<< (ostream& os, const SMeasResult& res)
    {
        static int resultCount = 0;
355     int i = 0;

        os << string(16, '=') << " Measurement Result " << ++resultCount << " " <<
            string(16, '=') << endl;
        if(SViComList<SSegmentMeas>::SLinkedObject* p = res.ListOfSegmentMeas.pFirst)
360     {
            os << "Channel Index" = " << res.dwChannelIndex << endl;
            os << "Center Frequency" = " << res.dCenterFreqInHz << " Hz" << endl;
            os << "Nominal Bandwidth" = " << res.dwNomBandwidthInHz << " Hz" << endl;
            os << "FFT Size" = " << res.dwFftSize << endl;
365     os << "CP Ratio" = " << res.dwCpRatio << endl;
            os << "Frame Rate" = " << res.dwFrameRateInHz << " Hz" << endl;

            i = 0;
            for(SViComList<SSegmentMeas>::SLinkedObject* p =
370 res.ListOfSegmentMeas.pFirst;

```

```

    p != NULL;
    p = p->pNext, ++i)
    {
375     os << "[Segment Measurement " << i << "]" << endl;
        os << "  Preamble Index    = " << (int)p->bPreambleIndex << endl;
        os << "  Segment              = " << (int)p->bSegment << endl;
        os << "  ID Cell                = " << (int)p->bIdCell << endl;
    }

380     i = 0;
    for(SViComList<SCir>::SLinkedObject* p = res.ListOfCirs.pFirst;
        p != NULL;
        p = p->pNext, ++i)
    {
385     os << "[CIR " << i << "]" << endl;
        if(p->pPowerDelayProfile)
        {
            os << "  RSSI                    = " << p->pPowerDelayProfile-
390 >fAggregatePowerInDBm << " dBm" << endl;
            os << "  CINR                     = " << p->pPowerDelayProfile->dCINR << "
dB" << endl;
        }
        os << endl;
395     }

    if(res.pDlfp)
    {
400     const SDlfp* pDlfp = res.pDlfp;
        os << string(16, '+') << " DLFP Result " << string(16, '+') << endl;
        os << "Channel Index          = " << res.dwChannelIndex << endl;
        os << "Subchannel Groups      = 0x" << hex << (int)pDlfp->bSubchannelGroups
<< endl;
        os << dec;
405     os << "Repetition Coding    = " << (int)pDlfp->bRepetitionCoding << endl;
        os << "DLMAP Coding          = " << (int)pDlfp->bDlmapCoding << endl;
        os << "DLMAP Length         = " << (int)pDlfp->bDlmapLength << endl;
    }

410     //
    // BCH demodulation results
    //
    if ( NULL != res.pPduResult )
    {
415     const SPduResult* pPDU = res.pPduResult;
        os << string(16, '-') << " PDU Result " << string(16, '-') << endl;
        os << "Channel Index          = " << pPDU->dwChannelIndex << endl;
        os << "Time stamp            = " << pPDU->dwTimeStampInMs << endl;
        os << "BTS ID                = " << pPDU->dwBtsId << endl;
420     os << "Preamble Index        = " << (int)pPDU->bPreambleIndex << endl;
        os << "Segment                = " << (int)pPDU->bSegment << endl;
        os << "Base Station ID      = 0x";
        os << hex;
        const int sizeofBSID = sizeof(pPDU->barBaseStationID) / sizeof(pPDU-
425 >barBaseStationID[0]);
        const BYTE* pBSID = pPDU->barBaseStationID;
        for(int i = 0; i < sizeofBSID; i++)
        {
            // High word first.
430     os << (int)((pBSID[i] & 0xF0) >> 4);
            os << (int)(pBSID[i] & 0x0F);
        }
        os << dec;
        os << endl;
435     os << "Sector ID              = " << (int)pPDU->bSectorID << endl;
        os << "Operator ID           = " << pPDU->dwOperatorID << endl;
        os << "PDU Type              = " << pPDU->ePDU << endl;
        os << "PDU Flags             = " << pPDU->dwFlags << endl;
        os << "PDU Bit Count         = " << pPDU->dwBitCount << endl;
440     // We don't display the bit stream pPDU->pbBitStream
    }

```

```

        return os;
    }
445 }
    /**
    Correctly interpret and display the Value field of a TLV structure basing on
    the Length field of the given TLV structure.
    @param[in] os    The ostream (normally, cout).
450 @param[in] tlv   Reference to the TLV struct to be dumped.
    @return    void.
    */
    void DisplayValueOfTlv(ostream& os, const STlv& tlv, const DWORD indent = 2)
455 {
    os << string(indent, ' ') << "Value          = ";

    if(tlv.pValue)
    {
460     switch(tlv.dwLength)
    {
    case sizeof(BYTE):
        os << (DWORD)(*(BYTE*)tlv.pValue);
        break;

465     case sizeof(WORD):
        os << *(WORD*)tlv.pValue;
        break;

470     case sizeof(DWORD):
        os << *(DWORD*)tlv.pValue;
        break;

    default:
475         os << "0x";
        os << hex;
        for(DWORD i = 0; i < tlv.dwLength; i++)
        {
            os << (DWORD)(*(BYTE*)tlv.pValue + i) << setfill('0') << setw(2);
480         }
        os << dec;
    }

    os << " (" << tlv.szValue << ")";
485 }
    os << endl;
}

/**
490 Helper operator to write TLV data to the console.
@param[in] os    The ostream (normally, cout).
@param[in] tlv   Reference to the TLV struct to be dumped.
@return    A reference to os.
*/
495 ostream& operator<< (ostream& os, const STlv& tlv)
{
    os << "  Type          = " << (int)tlv.bType << " (" << tlv.szType <<
    ")" << endl;
    os << "  Length        = " << tlv.dwLength << endl;
500
    switch(g_ePduForTlvDisplay)
    {
    const SDownlinkBurstProfile* pDlBurst;
    const SUpLinkBurstProfile*   pUlBurst;
505
    case CViComWiMaxInterfaceData::WIMAX_MAC_PDU_DCD:
        switch(tlv.bType)
        {
495         case 1: // Downlink Burst Profile
            pDlBurst = (const SDownlinkBurstProfile*)tlv.pValue;
            os << "    DIUC          = " << (int)pDlBurst->bDIUC << endl;
            if(pDlBurst->pTlv)

```



```

        {
            os << "    [Sub TLV]" << endl;
515         os << "    Type           = " << (int)pDlBurst->pTlv->bType << "
(" << pDlBurst->pTlv->szType << ")"<< endl;
            os << "    Length        = " << pDlBurst->pTlv->dwLength <<
endl;
            DisplayValueOfTlv(os, *pDlBurst->pTlv, 4);
520         }
            break;

        default:
            DisplayValueOfTlv(os, tlv);
525     }
        break;

    case CViComWiMaxInterfaceData::WIMAX_MAC_PDU_UCD:
        switch(tlv.bType)
530     {
        case 1: // Uplink Burst Profile
            pUlBurst = (const SUplinkBurstProfile*)tlv.pValue;
            os << "    UIUC           = " << (int)pUlBurst->bUIUC << endl;
            if(pUlBurst->pTlv)
535         {
                os << "    [Sub TLV]" << endl;
                os << "    Type           = " << (int)pUlBurst->pTlv->bType << "
(" << pUlBurst->pTlv->szType << ")"<< endl;
                os << "    Length        = " << pUlBurst->pTlv->dwLength <<
540 endl;
                DisplayValueOfTlv(os, *pUlBurst->pTlv, 4);
            }
            break;

545         default:
            DisplayValueOfTlv(os, tlv);
        }
        break;

550     case CViComWiMaxInterfaceData::WIMAX_MAC_PDU_MOB_NBR_ADV: // fall-through
        default:
            DisplayValueOfTlv(os, tlv);
        }

555     os << "    -----" << endl;

        return os;
    }

560 /**
    Helper operator to write L3 decoder result data to the console.
    You can extend this to display further L3 decoder result data.
    @param[in] os    The ostream (normally, cout).
    @param[in] l3res The SL3DecoderResultData struct to be dumped.
565 @return    A reference to os.
    */
    ostream& operator<< (ostream& os, const SL3DecoderResultData& l3res)
    {
        DWORD dwLen = l3res.dwStringLength;
570         os << endl << string(16, '-') << " PDU Decoded Contents " << string(16, '-')
<< endl;
        // You can show the complete description of the decoded PDU by the following
        commented-out lines.
575         if(l3res.pcStringPDU)
            {
                os << l3res.pcStringPDU << endl;
            }
        os << "PDU Type           = " << l3res.dwPDU << endl;

580         const SDlmap* pDlmap = l3res.pDlmap;
        if(pDlmap)
            {

```

```

585     os << "No. of IEs           = " << (int)pDlmap->bNoOfIEs << endl;
        os << "No. OFDMA Symbols   = " << (int)pDlmap->bNoOfDLSymbols << endl;
        os << "Base Station ID     = 0x";
        os << hex;
        const int sizeofBSID = sizeof(pDlmap->barBaseStationID) / sizeof(pDlmap-
>barBaseStationID[0]);
590     const BYTE* pBSID = pDlmap->barBaseStationID;
        for(int i = 0; i < sizeofBSID; i++)
        {
            // High word first.
            os << (int)((pBSID[i] & 0xF0) >> 4);
595         os << (int)(pBSID[i] & 0x0F);
        }
        os << dec;
        os << endl;
600     os << "Sector ID           = " << (int)pDlmap->bSectorID << endl;
        os << "Operator ID        = " << pDlmap->dwOperatorID << endl;
        os << "Frame Duration      = ";
        switch(pDlmap->bFrameDuration)
        {
605     case 1:
            os << "2.0ms";
            break;

        case 2:
610         os << "2.5ms";
            break;

        case 3:
            os << "4ms";
            break;
615     case 4:
            os << "5ms";
            break;

620     case 5:
            os << "8ms";
            break;

625     case 6:
            os << "10ms";
            break;

        case 7:
630         os << "12.5ms";
            break;

        case 8:
            os << "20ms";
            break;
635     default:
            os << "<Reserved>" << endl;
        }
        os << endl;
640     os << "Frame Number           = " << pDlmap->dwFrameNumber << endl;
        os << "DCD Count             = " << (int)pDlmap->bDCDCCount << endl;
        os << "No. OFDMA Symbols     = " << (int)pDlmap->bNoOfDLSymbols << endl;
        // Display the DL-MAP IEs.
        os << "DL-MAP IEs:" << endl;
645     for(SViComList<SDlmapIE>::SLinkedObject* p = pDlmap-
>ListOfDlmapIEs.pFirst;
        p != NULL;
        p = p->pNext)
        {
650         // Check if it is a burst profile IE
            if(p->bDIUC >= 0 && p->bDIUC <= 12)
            {
                os << " ---- Burst " << (int)p->bDIUC << " ----" << endl;
            }
        }

```

```

655 << endl;          os << " OFDMA Symbol Offset = " << (int)p->uIE.burst.bSymbolOffset
                    os << " Subchannel Offset = " << (int)p-
>uIE.burst.bSubchannelOffset << endl;
                    os << " Boosting = " << (int)p->uIE.burst.bBoosting <<
endl;
660                os << " No. OFDMA Symbols = " << (int)p-
>uIE.burst.bNoOfDLSymbols << endl;
                    os << " No. Subchannels = " << (int)p-
>uIE.burst.bNoOfSubchannels << endl;
                    os << " Repetition Coding = " << (int)p-
665 >uIE.burst.bRepetitionCoding << endl;
                }
            }
        }

670    const SULmap* pUlmap = l3res.pUlmap;
        if(pUlmap)
        {
            os << "UCD Count = " << (int)pUlmap->bUCDCount << endl;
            os << "Allocation Start Time = " << pUlmap->dwAllocationStartTime <<
675 endl;
            os << "No. OFDMA Symbols = " << (int)pUlmap->bNoOfULSymbols << endl;
            // Display the UL-MAP IEs.
            os << "UL-MAP IEs:" << endl;
            for(SViComList<SULmapIE>::SLinkedObject* p = pUlmap-
680 >ListOfUlmapIEs.pFirst;
                p != NULL;
                p = p->pNext)
            {
                // Check if it is a ranging IE
685                if(p->bUIUC == 12)
                {
                    os << " ---- Ranging " << (int)p->bUIUC << " ----" << endl;
                    os << " OFDMA Symbol Offset = " << (int)p-
>uIE.ranging.bSymbolOffset << endl;
690                    os << " Subchannel Offset = " << (int)p-
>uIE.ranging.bSubchannelOffset << endl;
                    os << " No. OFDMA Symbols = " << (int)p-
>uIE.ranging.bNoOfULSymbols << endl;
                    os << " No. Subchannels = " << (int)p-
695 >uIE.ranging.bNoOfSubchannels << endl;
                    os << " Ranging Method = " << (int)p-
>uIE.ranging.bRangingMethod << endl;
                    os << " Dedicated Ranging Indicator = " << (int)p-
>uIE.ranging.bDedicatedRangingIndicator << endl;
700                }
            }
        }

705    const SDcd* pDcd = l3res.pDcd;
        if(pDcd)
        {
            g_ePduForTlvDisplay = CViComWiMaxInterfaceData::WIMAX_MAC_PDU_DCD;
            os << "Config Change Count = " << (int)pDcd->bConfigurationChangeCount
<< endl;
710            // Display the TLVs.
            os << "DCD TLVs:" << endl;
            for(SViComList<STlv>::SLinkedObject* p = pDcd->ListOfTlvs.pFirst;
                p != NULL;
                p = p->pNext)
715            {
                os << *p;
            }
        }

720    const SUcd* pUcd = l3res.pUcd;
        if(pUcd)
        {
            g_ePduForTlvDisplay = CViComWiMaxInterfaceData::WIMAX_MAC_PDU_UCD;

```

```

725     os << "Config Change Count    = " << (int)pUcd->bConfigurationChangeCount
<< endl;
    os << "Ranging Backoff Start  = " << (int)pUcd->bRangingBackoffStart <<
endl;
730     os << "Ranging Backoff End    = " << (int)pUcd->bRangingBackoffEnd <<
endl;
    os << "Request Backoff Start  = " << (int)pUcd->bRequestBackoffStart <<
endl;
    os << "Request Backoff End    = " << (int)pUcd->bRequestBackoffEnd <<
735 endl;

    // Display the TLVs.
    os << "UCD TLVs:" << endl;
    for(SViComList<STlv>::SLinkedObject* p = pUcd->ListOfTlvs.pFirst;
740     p != NULL;
        p = p->pNext)
    {
        os << *p;
745     }

    const SMobNbrAdv* pMobNbrAdv = l3res.pMobNbrAdv;
    if(pMobNbrAdv)
750     {
        g_ePduForTlvDisplay = CViComWiMaxInterfaceData::WIMAX_MAC_PDU_MOB_NBR_ADV;

        os << "Skip-optional-fields = " << (int)pMobNbrAdv->bSkipBitmap << endl;
        if(!(pMobNbrAdv->bSkipBitmap & 0x01))
755     {
            os << "Operator ID          = " << pMobNbrAdv->dwOperatorID << endl;
        }
        os << "Config. Change Count = " << (int)pMobNbrAdv->
>bConfigurationChangeCount << endl;
        os << "Fragmentation Index = " << (int)pMobNbrAdv->bFragmentationIndex
760 << endl;
        os << "Total Fragmentation = " << (int)pMobNbrAdv->bTotalFragmentation
<< endl;
        os << "N_NEIGHBORS          = " << (int)pMobNbrAdv->bNNeighbors << endl;

765     // Display the Neighbors.
        os << "Neighbors:" << endl;
        for(SViComList<SNeighbor>::SLinkedObject* p = pMobNbrAdv->
>ListOfNeighbors.pFirst;
770     p != NULL;
        p = p->pNext)
        {
            os << "Length                = " << (int)p->bLength << endl;
            os << "PHY Profile ID          = " << (int)p->bPhyProfileID << endl;
            if(p->bPhyProfileID & 0x02) {
775     os << "FA Index                = " << (int)p->bFAIndex << endl;
            }
            if(p->bPhyProfileID & 0x08) {
                os << "BS EIRP                = " << (int)p->bBsEirp << endl;
            }
780     if(!(pMobNbrAdv->bSkipBitmap & 0x02)) {
                os << "Neighbor BSID          = " << p->dwNeighborBSID << endl;
            }
            os << "Preamble Index          = " << (int)p->bPreambleIndex << endl;
            if(!(pMobNbrAdv->bSkipBitmap & 0x04)) {
785     os << "HO Process Opt.          = " << (int)p->bHOProcessOptimization
<< endl;
            }
            os << "Scheduling Svc. Sup.    = " << (int)p->
>bSchedulingServiceSupported << endl;
790     os << "DCD CCC                = " << (int)p->
>bDCDConfigurationChangeCount << endl;
            os << "UCD CCC                = " << (int)p->
>bUCDConfigurationChangeCount << endl;

795     // Display the TLVs.

```

```

    os << "MOB_NBR-ADV TLVs:" << endl;
    for(SViComList<STlv>::SLinkedObject* pTlv = p->ListOfTlvs.pFirst;
        pTlv != NULL;
        pTlv = pTlv->pNext)
800     {
            os << *pTlv;
        }
    }
}
805
const SRngRsp* pRngRsp = l3res.pRngRsp;
if(pRngRsp)
{
    g_ePduForTlvDisplay = CViComWiMaxInterfaceData::WIMAX_MAC_PDU_RNG_RSP;
810
    // Display the TLVs.
    os << "RNG-RSP TLVs:" << endl;
    for(SViComList<STlv>::SLinkedObject* p = pRngRsp->ListOfTlvs.pFirst;
        p != NULL;
        p = p->pNext)
815     {
            os << *p;
        }
    }
820
const SRegRsp* pRegRsp = l3res.pRegRsp;
if(pRegRsp)
{
    g_ePduForTlvDisplay = CViComWiMaxInterfaceData::WIMAX_MAC_PDU_REG_RSP;
825     os << "Response" << " = " << (int)pRegRsp->bResponse << endl;

    // Display the TLVs.
    os << "REG-RSP TLVs:" << endl;
    for(SViComList<STlv>::SLinkedObject* p = pRegRsp->ListOfTlvs.pFirst;
        p != NULL;
        p = p->pNext)
830     {
            os << *p;
        }
    }
835
    os << endl << endl;

    return os;
840 }

/**
Worker function. The structure is pretty easy. In contrast to other sample
845 case that a problem arises, and these exceptions will be handled in the calling
method.
@param[in] rInterface A reference to the CViComWiMaxInterface to operate on.
@param[in] measConfig A reference to the measurement configuration struct.
@return void.
850 */
void doWiMaxMeasurement (CViComWiMaxInterface& rInterface, const SSettings&
measConfig)
{
    const SMeasResult* pRes = NULL;
855     DWORD const dwTimeOutInMs = 10000;

    // Setting up the measurement
    rInterface.GetBasicInterface().SetResultBufferDepth (
measConfig.ResultBufferDepth );
860     rInterface.SetFrequencyTable (
measConfig.ChannelSettings );
    if(measConfig.BchDemodulationSettings.sStartMeasurementRequests.dwCountOfPDUR
equests > 0)
    {
865         rInterface.SetBchDemodulationSettings (
measConfig.BchDemodulationSettings );
    }
}

```

```

    }

    rInterface.GetBasicInterface().StartMeasurement();
870
    // Getting measurement results
    for(;;)
    {
        if ( !_kbhit() )
875         break;

        try {
            pRes = rInterface.GetResult(dwTimeOutInMs);
            cout << *pRes;
880

            if ( NULL != pRes->pPduResult )
            {
                // Decode PDU and display the text
                SL3DecoderRequestData  sRequest;
885                sRequest.dwPDU        = pRes->pPduResult->ePDU;
                sRequest.dwFlags       = pRes->pPduResult->dwFlags;
                sRequest.dwBitCount    = pRes->pPduResult->dwBitCount;
                DWORD dwByteCount     = (sRequest.dwBitCount+7)/8;
                sRequest.pbBitStream   = new BYTE[dwByteCount];
890                memcpy( sRequest.pbBitStream, pRes->pPduResult->pbBitStream,
dwByteCount );
                // You can disable the 'pcStringPDU' filling in the result
                'SL3DecoderResultData' structure
                // by uncommenting the following line.
895                //sRequest.dwDecodingFlags = 0;

                const SL3DecoderResultData*  pL3data =
                rInterface.RetrieveTextForPDU( sRequest );
                cout << *pL3data;
900

                delete [] sRequest.pbBitStream;
                sRequest.pbBitStream = NULL;
            }
        }
        catch (const CViComError& err)
        {
            DUMP_VICOM_ERROR(err);
        }
    }
910

    // Stopping the measurement
    rInterface.GetBasicInterface().StopMeasurement();
    rInterface.GetBasicInterface().HasMeasurementStopped(dwTimeOutInMs);

915    S_WiMax_PDU_Request* pStartMeasPDURquest =
    measConfig.BchDemodulationSettings.sStartMeasurementRequests.ppDURquest;
    if(NULL != pStartMeasPDURquest)
    {
        delete [] pStartMeasPDURquest;
920        pStartMeasPDURquest = NULL;
    }
}

/**
925 * Sample program entry point.
*/
int main(int argc, char* argv[])
{
    cout << "Demo console application for WiMAX ViCom." << endl << endl;
930

    if(!GetArgOptions(argc, argv))
    {
        return 1;
    }
935

    CViComLoader<CViComWiMaxInterface, CViComLoader_TSMW> loader;

```

```

    try
    {
940         //
           // TSMW devices have to be connected using the IP address at which they
           // can be reached.
           //
           loader.ConnectTsmw(g_sAddr.c_str());
945         //
           // Assign the resources
           //
           CViComBasicInterface& rIfc = loader->GetBasicInterface();
950         // Use frontend 1 or 2 with up to 50% load for scanning and demodulation

           etTsmwFrontendId eFeScanner;
           etTsmwFrontendId eFeDemodulator;
           eFeScanner = (g_sFrontEnds[0] == '1') ?
955 CViComBasicInterfaceData::RF_FRONTEND_1 :
           CViComBasicInterfaceData::RF_FRONTEND_2;
           eFeDemodulator = (g_sFrontEnds[1] == '1') ?
           CViComBasicInterfaceData::RF_FRONTEND_1 :
           CViComBasicInterfaceData::RF_FRONTEND_2;
960         ResourceHandle resourceScanner =
           rIfc.GetTsmwResource(eFeScanner, 50);
           ResourceHandle resourceDemodulator =
           rIfc.GetTsmwResource(eFeDemodulator, 50);
965         doWiMaxMeasurement (*loader, getWiMaxMeasConfig(resourceScanner,
           resourceDemodulator));

           // Disconnecting the TSMW is done using the method shown below, not
           // Release() known from the TSMx related CViComLoader classes
           //
           loader.DisconnectTsmw();
           }
           catch (const CViComError& err)
975         {
           DUMP_VICOM_ERROR(err);
           }
           return 0;
           }
980

```

The interesting parts in the sample code are the Initialization (lines 870 - 894) of TSMW, the Measurement Configuration (lines 270 - 318), the Results Output using the overloaded operator << for scanner results (lines 328 - 416), L3 decode results output (lines 536 - 780) and TLV results output (lines 467 - 527).

9.3.1 Initialization Sequence

Initialization of the TSMW is done here with the IP address of the TSMW specified in the command line, or the default address (192.168.0.2) if none given. If this address has been changed, it must be adapted to the new setting or change the command line to make the sample application run properly.

9.3.2 Measurement Configuration

The measurement configuration is performed in the `getWiMaxMeasConfig()` function. It is configured to perform scanner measurement on frequencies (and the corresponding bandwidths) given in the command line, or a single frequency 2600 MHz (line 245) with bandwidth equals to 10MHz. The function also sets the demodulation CINR threshold value (line 75). Once the threshold is set, the program will only demodulate the scanned WiMAX signal when its CINR value is greater than the threshold value.

The scanner measurement is setup to run on front-end 1 and demodulation on front-end 2 each with a 50% load in the `main()` function (line 861).

In the main measurement loop (lines 808 - 842), the code processes all results as long as they are found within 10 seconds or unless a key is pressed.

9.3.3 Results Output

Result output is performed using two overloaded stream operator `<<`, one for `SMeasResult` and one for `SL3DecoderResultData`, and one for TLV contents. They dump the contents of the result to `stdout`, and also demonstrate how to access the elements inside the `SViComList`.

There are also some helper functions such as `GetArgOption()` (lines 235 – 261) to process the program arguments and set the corresponding parameters (Center Frequency, Bandwidth and so on) accordingly. The usage can be found inside the `RunSample.bat` batch file that comes along with the sample source code.

9.3.4 Command Line Arguments

-ADDR=<IP address of TMSW>

-CF=<Center frequency> (Each **-CF** specified defines a new channel)

-BW=<Bandwidth> (Corresponds to the above center frequency specified)

-CINR=<CINR threshold value> (For all channels)

-FE=<Front-End specification> (For all channels. The parameters followed can be any of the following four: 11, 12, 21, 22. The first digit specifies which front-end is used for scanning and the second specifies which front-end is used for demodulation. For example: **-fe=12** specifies using front-end 1 for scanning and front-end 2 for demodulation)

The major command line inside the batch file is as follows:

```
SampleForViComWiMax_cmd_Release.exe -addr=192.168.0.1 -
cf=2600000000 -bw=10M -pdu=2 -pdu=3 -pdu=1 -cf=2345000000 -bw=5M
-pdu=2 -pdu=53 -cinr=15.1 -fe=12
```

Which retrieves measurement results from TSMW with IP address 192.168.0.1, scans at center frequency 2.6 GHz with bandwidth 10MHz for DL-MAP (PDU type: 2), UL-MAP (PDU type: 3) and DCD (PDU type: 1), and another center frequency 2.345 GHz with bandwidth 5M for DL-MAP (PDU type: 2) and MOB_NBR-ADV(PDU type: 53). The CINR threshold for the scanning is set to 15.1dB.

10 ViCom RF Power Scans

This chapter describes how the power frequency scan API is used to control one or more TSMx device to measure the spectrum of multiple frequency ranges in a high-speed fashion.

The first section gives insight into how the RF Power Scanner library and its firmware perform measurements with a TSMx device. It is crucial for understanding the remaining sections to know how the results are created from the raw measurement data that the TSMx provides.

In the second section a rough overview of how the power frequency scan API is designed and how the scan values are calculated is given. This explains the configuration details and the logic behind the post-processing units used to prepare a frequency spectrum.

Section Sample Application demonstrates the usage of the packaged sample application. The design of the demo application is tightly coupled to the API, so once one can use this demo application, programming the API becomes easy.

A small sample program is given in section Sample to provide a starting point for your own experiments with the ViCom RF Power Scanner.

The chapter concludes with a small FAQ section in that some common pitfalls are discussed.

10.1 Measuring with the RF Power Scanner

The TSMx series is not directly designed to be a frequency spectrum analyzer. Such functionality is implemented in the TSMx in software and firmware. Several software modules are available to perform typical tasks of a frequency spectrum analyzer.

Most of the time, measurements within the TSMx will be done asynchronously to user requests, which has certain impacts on the design of the API. One such measurement from a lower frequency to higher one is called sweep. The TSMx device performs sweeps at a constant rate.

To generate a frequency spectrum from raw TSMx measurements, at least one sweep must be performed. With one or more sweeps, one user request for a frequency spectrum can be satisfied. In the simplest case, measurement data would directly translate to the result. However, this case may only occur rarely.

Most of the time, the result will be created from a set of sweeps. Since there is no unique algorithm to perform such a transformation, the ViCom RF Power Scan interface provides many configuration options.

Data for a frequency spectrum may be requested when one of the conditions below is valid:

- No sweep data is available at the TSMx. In that case, the response has to be postponed until data is available. When another result is requested while the first one is waiting, the second request will not be answered separately. So one important consequence of this is that there might be fewer results that requests.

- One sweep is available. This simple case can be translated easily into a result. Nevertheless, it's not that simple, the frequency spectrum might be converted into a channel spectrum.
- At least two sweeps are available in the TSMx cache. In such a case, all the raw measurements have to be combined into one spectrum. There are several possible ways to do this, so the user can choose how this actually done. For example, data for a single frequency can be the maximum of the set of power values.

All these scenarios are shown in the figure below. On the left side, 4 user requests are depicted in their order of occurrence. An internal timer is started to issue all these requests in regular intervals. On the right side, sweep data is coming from the TSMx and stored in the data cache. The measurement data is processed to derive a result for a request then.

Both requests and sweep measurement data have a timestamp assigned to them in the figure. Timestamps are marked as t_x , where a measurement at t_{x+1} occurred at a later time than one done at t_x .

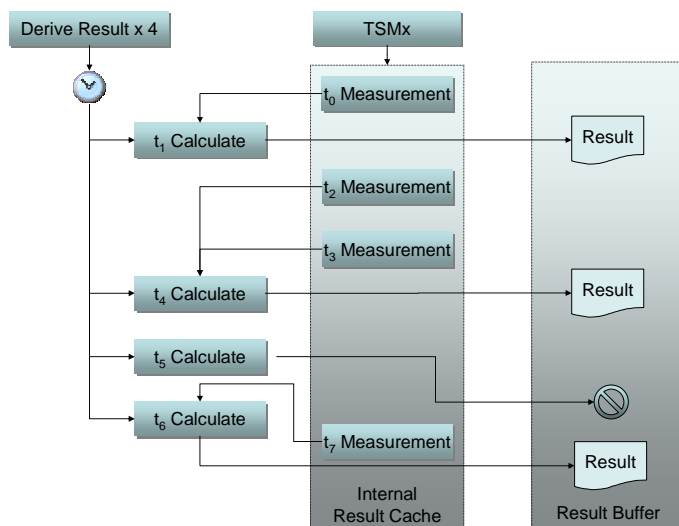


Figure 49 Request processing scheme over time

The situation depicted in the figure above is handled like explained below:

When the processing of the first request is started, a single sweep is available in the TSMx cache. The measurement data is used directly as result. This is always the case for the first request: If there is any data in the cache, all the data is used to satisfy the need of the first request. If there is no such data, the processing of the request is delayed until data arrives or a new request is issued.

The second request is issued after two sweeps have been stored in the result cache. These two measurements are combined into a result, which is made available to the caller.

For some reason, no measurement data arrives after the third request has been issued, and no data can be found in the cache. Since no data arrives in the internal result cache until the fourth request is activated by the timer, no result is created for the third request.

The measurement at t_7 is used as input for the final request. If a request cannot be satisfied with data from the data cache, the processing is delayed until one sweep has been done and data is available again. If there is a new request put into the queue, the old request is cancelled and no result will be generated.

10.2 Architecture and Functionality of the RF Power Scanner

This section describes the architecture of ViCom RF Power Scanner Interface, along with the details of how the different post processing units work.

The main goal during the design phase was to have a simple and efficient way to make the TSMx device perform multiple different measurement tasks in parallel. Another requirement was to provide the developer with a highly configurable system that can be used to get exactly those results one would expect without doing complex calculation manually.

All the abstraction layers described below were introduced to meet these goals. The next section describes the system components and abstraction layers. This is followed by a detailed analysis of what kind of post-processing capabilities are offered.

10.2.1 System Layers

As mentioned above, one design goal was to make parallel measurements easy and manageable. So there had to be some kind of middle tier between the physical TSMx device and the object that can be used to retrieve measurements.

For each physical TSMx unit, multiple logical devices in the software can exist. These logical devices are called RF Power Scanners. They all run in their own designated thread, in which they process the incoming results from the TSMx. Such a RF Power Scanner has some basic configuration attributes, among them the frequency range that is scanned and the derived result buffer.

Once a RF Power Scanner object has started communication with the TSMx, it is able to process the incoming measurement results. To prepare the raw measurement data for analysis, the developer must use a so-called derived result to request data.

Such a derived result monitors the data in the RF Power Scanner over some defined time interval. It extracts and processes those measurement data that is required to fulfill the request constraints. Once a raw measurement data (called sweep) has been processed, the result is stored in the RF Power Scanners result buffer. From there you can get them via the ViCom API asynchronously.

Since there might be different measurement task at one time, one RF Power Scanner can handle up to ten derived result requests in parallel.

The figure below shows this system architecture and the interaction. Note that you can also have more than one RF Power Scanner running in the so-called Sweep Worker thread to communicate with a single TSMx.

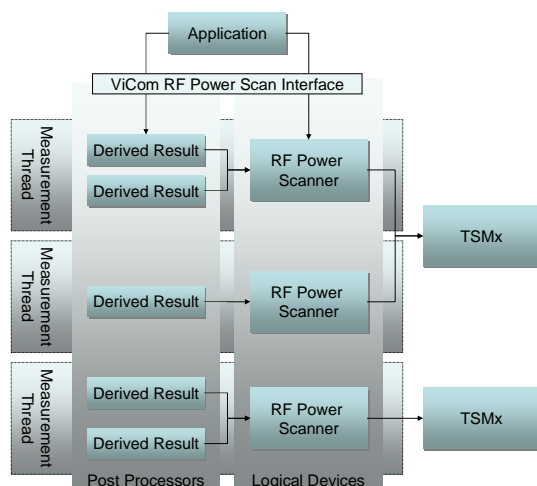


Figure 50 RF Power Scanner Architecture

10.2.2 The Post Processor Chain

In the previous section, the calculation of the derived results was described as a way to convert the raw measurement data from the TSMx to a more convenient presentation form. The details how to configure that post process are explained below.

One important part of data preparation is reduction. Since it is not useful in most cases to use each of the incoming values for analysis, it is crucial to reduce the amount of data to a manageable size. For example, most GUI applications won't be able to display all measured power values in the frequency range from 100 MHz to 1 GHz, using the native resolution of the TSMx of 12.8 KHz. This would result in approx. 70000 values, which is difficult to show using about 1000 pixels width of modern screens.

The post-processing step is designed in a modular way so that processing is split up in small units. The processing units work in a pipeline: Each element takes the input of the previous element, applies its own filter and calculation mechanisms, and provides the result to the next element.

This design was chosen because post-processing can be configured in many ways. For example, one use case is to aggregate data into channels. Based on that result, one might want to know what the strongest transmitter is in such a channel list. In another step, all the raw values shall be analyzed to also find adjacent interferers. The channel calculation is the same in both cases, so it is convenient to have that configuration shared among both use cases.

To perform these things with one single device the approach described above was designed. Refer to the figure below to get a feeling for how the post-process calculation takes place.

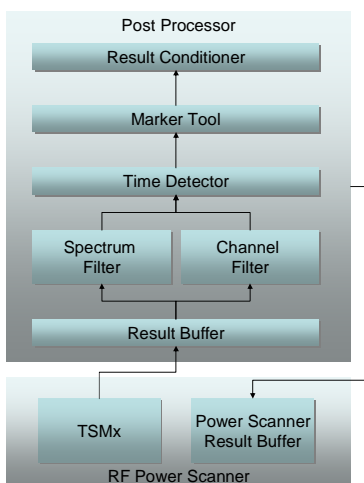


Figure 51 Post Processor Chain

As can be seen, there are seven different processing modules available. The one that starts processing is the sweep result buffer, the final result is returned from the result conditioner. These results are stored in the RF Power Scanner result buffer. To simplify handling, the API supports multiple equally configured post processes to be performed in one row.

One important detail to notice is that the spectrum filter and channel filter unit are mutually exclusive. Exactly one of both can be active. If both are enabled using the API, only the spectrum filter will perform its work, the channel filter won't be used.

The units and the calculations they perform are now described in the order of processing sequence.

10.2.2.1 Sweep Result Buffer

The result buffer, as the name implies, is used to save the incoming TSMx results. The data structure where these results are stored in provides the subsequent filters to find appropriate values faster than searching in a complete list of values.



This is not the same result buffer that stores the post processing results in the RF Power Scanner. This one stores the raw measurement data gathered from the TSMx.

The results are only stored for some amount of time. After a measurement sample reached its lifetime, it is purged when the next sample arrives.

10.2.2.2 Spectrum Filter

The task of this module is to select only frequencies from the result buffer that fit into a given sub-range and adapting the samples to a given number of result values. The sub-range must be contained within the RF Power Scanner sweep range interval.

The number of input values available in the raw sweep is: $\frac{f_{\max} - f_{\min}}{0.0128}$ where the frequencies are given in MHz. For a sweep scan range of 100 MHz, there will be approx 7812 values in the result buffer.

In many cases the output of the frequency selection does not directly map to the number of results you might need. The spectrum filter unit helps you to get the number of samples you need in your application.

If the number of measurements selected from the raw data does not fit to the number of results required, there are two possible approaches: Either there are less measurements than results, in which case the data has to be interpolated. Or the reduction in the previous modules on the data set was not restrictive enough, so the data has to be compressed once more.

In the API, the number of result samples is called `dwCountOfDisplayLines` and is set in the `SSpecificParameter` structure.

Data Interpolation

If more display lines are requested than available, the data has to be interpolated. To calculate power values for frequencies that don't directly match the native resolution, the power value is the result of a linear interpolation.

If the nearest lower frequency is f_{\min} and the nearest higher frequency is f_{\max} , the power for the requested frequency f is calculated like:

$$p(f) = p(f_i) + (p(f_{\max}) - p(f_{\min})) \frac{f - f_i}{f_{\max} - f_{\min}}$$



Note that the power values are converted to Watt internally before the interpolation can be performed, since the input values must be linearized. This might result in small numerical errors when transforming the power values from one representation into another and back.

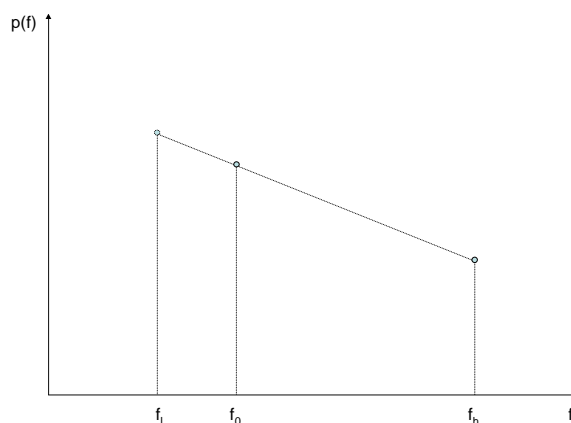


Figure 52 Linear Interpolation

Data Compression

This is used to reduce the amount of data from the previous units. Most of time, this will be necessary. The input data set is divided in the number of classes configured with the `dwCountOfDisplayLines` member. For each class, one result value will be calculated.

In order to perform a meaningful compression, you have the choice between three types of how to find the representing value for one data class:

- Max Peak: This uses the maximum value in the class.
- Auto Peak: Returns both the minimum and maximum value of a class, so you get twice the number of results requested
- RMS: The Root-Mean-Square of all values. To calculate this value, the power values stored in dBm are converted to Watt temporarily (see comment above).

The figure below shows that mechanism: The small dots represent a single power value for one of the native frequencies. The twelve native measurements from the sweep data have to be compressed to three frequencies (classes). The Max Peak selector would then choose the native measurement marked in red, whereas the Auto Peak would additionally add the measurement samples marked in a light yellow. The RMS would calculate the average of the native measurements, which are shown as bigger circles in the figure.

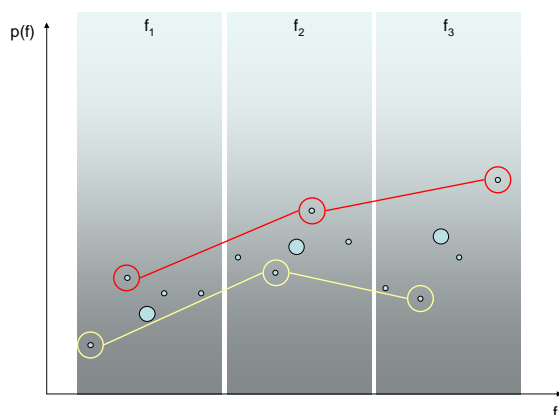


Figure 53 Power value compression

The decision which algorithm has to be applied is done by the spectrum filter itself. The only thing to configure is the number of result lines which are desired and the frequency interval borders.

10.2.2.3 Channel Filter

The channel filter can be used to calculate power values for a set of frequency channels. That means that the sample values that fall into the bandwidth are used to calculate the power value of the channel, although it is not required that only values of the bandwidth are used. Under some circumstances, it might be useful to also consider the adjacent power values for the bandwidth calculation.

The channels can be defined in an arbitrary way, even as a mixture of evenly spaced sub-sequences and freely defined ones. Each channel is specified by its centre frequency and its span. Optionally, multiple equidistant channels can be defined by specifying a repetition count.

For the rest of the section it is important to know that the TSMx device has a native frequency resolution of 12.8 KHz. These sample values cannot be used directly for the calculation of a channels power value (except for very specific configurations).

The central element of the algorithm to calculate the overall channel power is a power scale function for the measurement samples. It is used to control how different raw measurements contribute to the overall channel bandwidth.

The power scale function is always symmetrical and is defined in equidistant steps from the center frequency to some lower/upper limit. The limit is implicitly defined by the number of equidistant points and the spacing between those points. The calculation of the channel bandwidth power p is done as follows:

$$p = c \sum_{i=0}^N p(f_c + \Delta f) s(i)$$

where

- f_c is the center frequency of the channel
- Δf is the frequency span
- $s(i)$ is the i^{th} element in the power scale function. All the values of the power scale function must be within $[0; 2]$
- c is a correction factor to equalize the impact of the weights on the measurement samples
- N is the number of points that define the power scale function

This power scale function is the base for a channel filter. In the figure below, the basic principle is depicted. All the values explained above are shown there again. It is noteworthy that the channel bandwidth for a channel is defined by the scaling function. It can be calculated as $2 * N * \Delta f$.

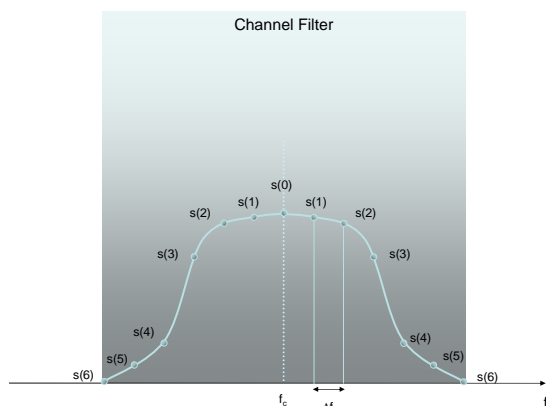


Figure 54 Channel Filter Definition

For example, if center frequency is 935.2 MHz and there are 4 points, spaced by 200 KHz, the power scale function is applied to the incoming spectrum at frequencies 934.6 MHz, 934.8 MHz, 935.0 MHz, 935.2 MHz, 935.4 MHz, 935.6 MHz and 935.8 MHz.

This approach can be used to make the center frequency dominant, for example, by setting the weight function to 2 at the center and to low values for the rest. Another use case is that channels might overlap and consider parts of the neighbour channels as well.

Usually, the (frequency; weight) pairs that define the power scale function don't map directly to the native measurement values. In such a case, the required values will be interpolated using the two nearest neighbours and a linear interpolation.

The sum of all the weighted values is then multiplied by a correction factor to reduce the power to a meaningful value. This algorithm is visualized in the figure below. In the figure, raw measurement samples are shown as small dots. Marked with a background is the channel for which the bandwidth power shall be calculated. The power scale function is shown in light blue.

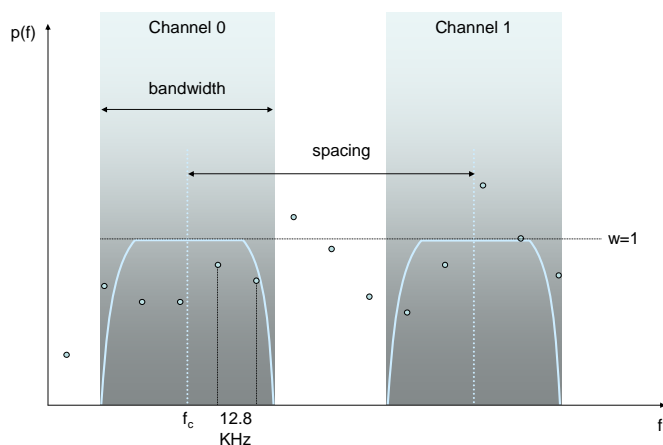


Figure 55 Calculation of Channel Power for a Channel Sequence

The channel filter is applied to every defined channel to calculate its power. The resulting number of values is therefore the number of channels requested. Channels can be defined in so-called sub-sequences. One sub-sequence is defined by a center frequency, the number and the spacing between the remaining center-frequencies.

10.2.2.4 Time Detector

Since there might be multiple sweeps that have to be combined into one result, the kind of aggregation in terms of time has to be specified as well. The choice made here also influences the number and layout of the result.

As for the spectrum filter, the central problem is how to find a representing value from a set of values for a frequency gathered over time. Compared to the frequency detector, the different values from which one must be chosen have the same frequency.

The following algorithms are available (in brackets numerical values are given to use them in the sample application):

- Max Peak (0): The maximum value from all power values inside the data set for the frequency/channel.
- Auto Peak (2): Maximum and minimum value are used. So the result of the request will be twice the number of values from the spectrum/channel filter.
- Min Peak (3): The minimum value is used analogue to the Max Peak.
- RMS (1): The root-mean-square value is calculated of all input values to retrieve one output value.



The original power values have to be converted into Watt internally to do this calculation. This conversion might result in small numerical errors.

- All (5): All values are used and returned as a list of measurement values in one chunk. So if data is available from more than one sweep, the result consists of a series of single results. For example, the output of a frequency detector post-process might be doubled etc. when two sweeps were made and all raw data shall be used.
- Histogram (4): The RF Power Scan interface provides the possibility to return a histogram, where for each (frequency; power value) or (channel; power value) the number of occurrences in a set of measurements is returned. The result has to be interpreted as a two dimensional array, where the columns reflect the frequencies or channels, and the rows specify the power value (in integer values). Each row covers a certain range of power values. The range for a column is defined as follows:
 - For row index 0, all power values smaller than $f_{MinPowerValueInDBm}$ are counted.
 - For row index $i = 1$ to $w_{MaxPowerInteger} - 1$, the power value is in the range $[f_{MinPowerValueInDBm} + i * f_{PowerResolutionInDB}; f_{MinPowerValueInDBm} + (i + 1) * f_{PowerResolutionInDB}]$
 - For the row index $w_{MaxPowerInteger}$, all power values that are greater than or equal to $f_{MinPowerValueInDBm} + w_{MaxPowerInteger} * f_{PowerResolutionInDB}$ are considered.

The example below shows a potential result of this function. As input four channels were defined, the minimum level set to -120 dBm and resolution to 5 dB. The maximum power integer was set to 4.

Table 7 Sample HistogramTable 10-8:

	935.2 MHz	935.4 MHz	935.6 MHz	935.8 MHz
0: $p < -115$ dBm	8	6	0	3
1: $-115 \leq p < -110$	2	6	1	3
2: $-110 \leq p < -105$	1	0	4	2
3: $-105 \leq p < -100$	1	0	7	4
4: $p \geq -100$ dBm	0	0	1	0

In the figure shown above, twelve sweeps were used to generate the resulting histogram. In these twelve sweeps, a value between -115 dBm and -110 dBm was measured 6 times for frequency 935.4 MHz. Similar, the frequency 935.6 MHz was 4 times in the range -110 dBm and -105 dBm. The maximum power integer was set to 4. So all values greater than -100 dBm will be put into the row containing the maximum power integer. For frequency 935.8 MHz, the histogram shows that there was a power value less than -115 dBm measured three times.

10.2.2.5 Marker Tool

The marker tool helps when it comes to select the frequency that provided the maximum power value in a set of frequencies. With this it is an easy task to report the strongest power value and frequency.

If the marker tool is enabled, it reports the result of the marker evaluation as part of result.



The measurement values are also changed so that they can't be interpreted as mentioned above. After the marker has been applied, they reflect the values of the marker frequency in each different raw sweep, after the frequency detector or the channel filter processing has been done.

For example, if there were three sweeps made by the TSMx, the input data might look similar to:

Table 9 Sample for marker result values

1. Sweep	-61.9	-77.1	-65.4	-71.4	-64.2	-62.2
2. Sweep	-62.3	-79.1	-69.3	-71.3	-61.9	-62.1
3. Sweep	-63.8	-78.8	-69.2	-71.9	-62.4	-61.5
RMS	-62.6	-78.3	-67.9	-71.5	-62.8	-61.9

The maximum is calculated based on the result of the previous processing steps. In this case, we assume that the Time Detector is configured to calculate the root mean square of all input samples. The highest frequency would result in the maximum power here. As marker result, this maximum of -61.9 dBm would be returned then as special marker result. The standard result structure would contain -62.2 dBm, -62.1 dBm and -61.5 dBm.

10.2.2.6 Result Conditioner

The result conditioner is a utility to convert the resulting power values into a specific layout. Therefore it provides several options:

Minimum Power Level and Resolution

The final results are returned in a WORD-Array. The unsigned integer numbers stored there have to be set in relation to a minimum threshold and have been shifted to make the minimum resolution that shall be available an integer.

For example, a resolution of 0.1 and a base level of -120, the value 234 means $-120 + 234 * 0.1 = -96.6$ dBm

Histogram Axis Definition

The maximum value of the power dimension when a histogram is calculated can be set in this structure. That means, that the size of the histogram is number of frequencies/channel * (maximum power value + 1).



Note that this value is specified as integer. The calculation of the dBm value is the same as mentioned above, so you can calculate that value by using the inverse function:

$$wMaxPowerInteger = (p[dBm] - fMinPowerValueInDBm) / fPowerResolutionInDB.$$

Overflow Representation

If an overflow occurs while the measurement is done in the TSMx, this can be signaled in the result. The least significant bit in the result can be used as a flag indicating whether the value has been the result of some overflow (if set to true) or if not.

An overflow might arise in the A/D converter, for example. The TSMx can be configured to react in different ways in such a situation. Refer to the reference for more details on overflow handling.

10.3 Sample Application

As part of the ViCom API delivery is a dialog-based sample application provided. This helps to get a first feeling of how to use the ViCom Interface for RF Power Scans, since the applications user interface directly reflects the entities available in the API. See the figure below for a sample.

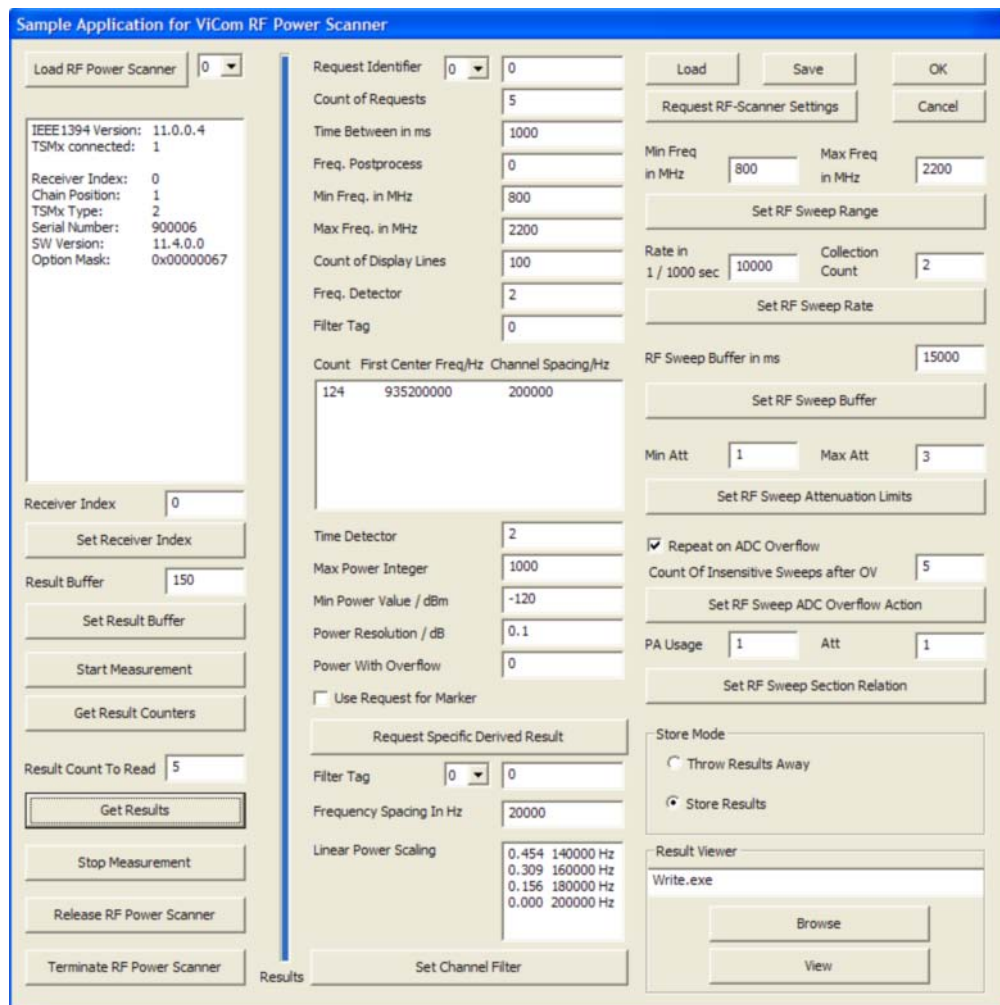


Figure 56 RF Power Scan Sample Application

As the figure above shows, the dialog is divided in three major columns. The left-most column manages the different logical receivers. Each receiver stands for one of the connected TSMx devices. Multiple logical receivers can be connected to one physical TSMx. You can configure up to 32 logical receivers. The column in the middle contains controls for managing the actual result configuration. Here you can specify how a concrete measurement shall be made. For one receiver, there can be up to 32 different request configurations. The settings here have to fulfill the constraints made in the right column. The right-most column controls general settings of the receiver and the application. Once you have set up a configuration, you can save and restore it here.

10.3.1 A Walk-Through Example

The general workflow when using the application is as follows:

1. Load receiver(s)
2. Read device settings
3. Change them to your needs and update them in the device
4. Start the measurement
5. Define specific requests
6. Fetch results
7. Stop measurement
8. Unload receivers(s)

Steps 3 to 7 can be repeated various times and with different settings.

10.3.1.1 Step 1: Load Receiver(s)

In this walkthrough, we assume that one TSMx device is connected to the computer. We want to have two logical receivers, one scanning below 1GHz, the other in the UMTS frequencies. Therefore, we load two receivers by first clicking one the “Load RF Power Scanner” button, then changing the combo box left to it the number 1 and click the button a second time. Each time the button is clicked you should see the text field below the button getting filled with information on the available scanners after some time (approx. 20-30 seconds).

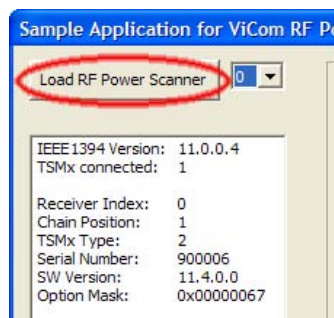
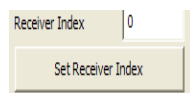


Figure 57 Load RF Scanner

In this case, we assume that only one TSMx unit is connected to the PC. If you have more than one such device, you can configure the application to use the second device as well. Each of the logical receivers loaded can be mapped to one of the available physical units by changing the Receiver Index field to the according value and then pressing “Set Receiver Index” (below the list of available devices).



The combo box left to the “Load RF Power Scanner” button controls to which logical receiver all the changes made in the rest of the GUI apply to. If you change the value in here, the control values also update accordingly, except for the settings in the right-most column.

10.3.1.2 Step 2: Reading the Device Settings

Before you change the general configuration of a logical receiver, you should first retrieve the current settings from the ViCom interface. This can be done by clicking the button “Request RF-Scanner Settings”. After the operation has finished successfully, the controls in the right column of the dialog should reflect the values from the logical device.

Figure 58 Sweep Settings section

In this section, the following settings can be changed:

- The sweep range, i.e. the interval for which derived results can be requested. That influences the measurement speed, so this has to be set carefully when speed is a critical factor.
- The sweep rate, which can be set to a slower value than the theoretical maximum if different measurements shall be made in parallel.
- The time how long sweep data is kept in the sweep result buffer before samples get invalid
- Attenuation settings
- Error handling strategy in case that the A/D converter detects an overflow

Note that these settings exactly match the values from the structures

- SRfSweepRange
- SRfSweepRate
- SRfSweepBuffer
- SRfSweepAttenuationLimits
- SRfSweepAdcOverflowAction
- SRfSweepSectionRelation

stored in the CViComRFPowerScanInterfaceData::SSettings structure.

10.3.1.3 Step 3: Set parameters

Before a changed value is set in the logical device, the related “Set” button below the input fields must be pressed to transfer the value into the related ViCom object. For our example, we set the Sweep Range of our first device from 100 to 1000, and for the second to 1800 to 2200. The minimum value for a frequency is 80 and the maximum 3000 (If you have a TSML CW device, the upper limit is 6000).

10.3.1.4 Step 4: Start Measurement

Once the common settings are set as desired, you can put the logical power scanner object into measurement state. Note that you have to do this for every instance. Once you did this, you cannot change common settings until you stop the measurement.



After you started a measurement, the mapped TSMx already scans all frequencies repeatedly. Nevertheless, you won't see any measurement results, unless you do specific measurement value requests.

10.3.1.5 Step 5: Define Specific Values

For each logical receiver, you can define up to ten different derived results from the scanning process. Each of these derived results can be requested in parallel. Such a result is called derived because the raw measurement values from the TSMx can be post processed to filter, reduce and prepare the measurement results in a more convenient form (refer to section 10.2.2.)

Request Identifier	0	0
Count of Requests		1
Time Between in ms		1000
Freq. Postprocess		0
Min Freq. in MHz		800
Max Freq. in MHz		2200
Count of Display Lines		100
Freq. Detector		2
Filter Tag		0

Figure 59 General Derived Result Settings

The figure above shows some of the settings you can specify for a derived request. Each different post-process is identified by the number shown in the combo-box. For our example, we create two request identifiers for the first logical device. Make sure that both the Power Scanner Identifier and the Request Identifier are set to zero. Enter the values 2, 1000, 0, 100, 500, 5 into the fields Count of Requests, Time Between in ms, Freq. Postprocess, Min/Max Freq. in MHz and Count of Display Lines. Press “Request Specific Derived Result” button to tell the scanner to record measurements and prepare them.

Switch the request identifier to 1 and enter 10, 100, 0, 600, 700, 20 in the fields mentioned above. Again, press the “Request Specific Derived Result” button.

10.3.1.6 Step 6: Fetch and View Measurement Results

Once you send a request to the TSMx using the “Request Specific Derived Result” button, an internal timer is configured to issue the number of requests in the specified time interval. This is done until the number of results has been processed or the measurement is stopped.



Note that you have enough space left in the result buffer to store all results from the requests you start at one time. To check how much space is already occupied, use the “Get Result Counter” button. The upcoming message box shows you how many results have been stored and how many have been thrown away already, if any.

You can change the size of the result buffer only if the logical device is in idle state (i.e. has not been started yet or stooped again).

As described in the architecture section before, the results from the post-processes are stored in a result buffer inside the memory. The sample application provides a way to retrieve values from that buffer (in FIFO order) and store them in a file. The output file ViComMeasurements.txt is stored in the Application\Logfiles directory.

Make sure to mark the radio button “Store Results”. Then put 10 in the text field next to “Result Count to Read” and press “Get Results”. The system then fetches ten results from the result buffer and writes them into the file mentioned above.

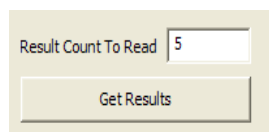


Figure 60 How to fetch results

If there are not enough results in the buffer to satisfy the request, the demo application waits for five seconds if other records arrive. If not, a message box is shown.

Any result that is available and requested will be written into the file. Each single record looks similar to the one shown below:

```
MeasResult:
dwPcTimeStampInMs = 12192898
  SpecificParameters:
    wRequestIdentifier = 0
    dwCountOfRequests = 2
    dwTimeBetweenRequestsInMs = 1000
    eFreqPostProcess = 0
      dMinFrequencyInMHz = 100.000000
      dMaxFrequencyInMHz = 500.000000
      dwCountOfDisplayLines = 5
      eFreqDetector = 2
    etTimeDetector = 2
    SPowerValueFormatSpec:
      wMaxPowerInteger = 1000
      fMinPowerValueInDBm = -120.000000
      fPowerResolutionInDB = 0.100000
      ePowerWithOverflowFormat = 0
      bUseRequestForMarker = 0

dwCountOfPowerWithOverflowValues = 10
pwPowerResultValues:
  581  0  312  0  250  0  268  0  249  0
bOverflowIndicator = 0
```

Figure 61 Output Example

The result contains from top to bottom the following information:

- When did the response arrive (in milliseconds after the start of the TSMx).
- The configuration details that were used to create that result
- The number of result values (`dwCountOfPowerWithOverflowValues`)
- The result values. In this case, we have five maximum/minimum pairs. The first pair (581, 0) is the one associated for `dMinFrequencyInMHz`, the last one for `dMaxFrequencyInMHz`. In between this interval, frequency values are placed in equidistant steps.

The maximum power can be calculated from the value 581 by multiplying the value with `fPowerResolutionInDB` and adding the result to the value of `fMinPowerValueInDBm`. In this sample, the power of the first value would therefore be $581 * 0.1 - 120 = -61.9$ dBm.

- The overflow indicator tells whether an overflow occurred during calculation of this result or not. If so, the values might not be the real measurement values, according to what was specified in the common settings of the power scanner before.

10.3.1.7 Step 7: Stop Measurement

To stop the logical device from recording raw measurement data, press the “Stop Measurement” button. After the receiver has been stopped, it is possible to change the common settings again.

You can now configure the logical devices again to perform some different measurements, or you can exit the application. Before the application is quit, the receivers should be unloaded to properly clean up acquired resources.

10.3.1.8 Step 8: Unload Receivers

Make sure to unload all logical receivers that were used using the “Release RF Power Scanner” button. If you encounter a problem during the release procedure, try to use the “Terminate RF Power Scanner” button. The ViCom API then forces a deletion of the `ViComRFPowerScanInterface` object.

10.3.2 Sample Project Organization

The application can be found as source code bundled in a Microsoft Visual Studio .NET 2003 project and solution in the sub directory `SampleForViComRFPowerScan` of the ViCom installation directory.

If you load the project/solution into Visual Studio, its completely ready to build. Once the project has been compiled, the resulting exe-file can be found in the Application directory of your installation (this applies to both Debug and Release configuration).

The project is organized as MFC dialog-based application. Therefore, the central class where all actions are performed is the `CSampleForViComRFPowerScanDlg` class in the `SampleForViComRFPowerScanDlg.cpp` file. In this class, the most important parts can be found in the `OnBnClickedXXX` methods. The XXX is a name similar to the caption of the button in the GUI, so it should be easy to find the place where you want to look at.

10.4 Sample

After all architectural and user concept discussions above, this section gets hands-on the code. The sample provided here lacks a sophisticated error handling strategy and design, but makes clear which steps have to be done to control the TSMx.

Below is the listing of the sample application. Right after the code the structure is explained and the most important pieces are discussed.

```

#include "stdafx.h"

#include <windows.h>
#include <iostream>
5
#include "ViComRFPowerScanInterface.h"
#include "ViComLoader.h"

using namespace std;
10

/*****

#define CHECK_VICOMERROR( err ) \
15     if ( err.GetErrorCode() > 0 ) { throw err; }

/*****

#define TO_DBM(v) \
20     tRequestParams.PowerValueFormatSpec.fMinPowerValueInDbm + \
     v * tRequestParams.PowerValueFormatSpec.fPowerResolutionInDB

/*****

25 void RunViComRfPowerScanSample()
{
    CViComError cError;
    CViComLoader< CViComRFPowerScanInterface > cRFPowerScanIfLoader;

30     //
    // First thing is to load the application dll and to initialize the TSMx
    // device for communication.
    //
    cout << "Loading TSMU...";
35     cRFPowerScanIfLoader.LoadTsmx( cError );
    CHECK_VICOMERROR( cError );
    cout << "ok\n";

    CViComRFPowerScanInterface* pcRFPowerScanner =
40     cRFPowerScanIfLoader.GetpInterface( cError );
    CHECK_VICOMERROR( cError );

    //
    // Now the TSMx is configured for the needs of this example. There
45     // is the result buffer size that is set, and the sweep interval
    // is specified.
    //
    CViComBasicInterfaceData::SResultBufferDepth tResultBuffer;
    tResultBuffer.dwValue = 128;
50     pcRFPowerScanner->GetBasicInterface().SetResultBufferDepth( cError,
    tResultBuffer );

    CViComRFPowerScanInterfaceData::SRfSweepRange tSweepRange;
    tSweepRange.dMinFrequencyInMHz = 935;
55     tSweepRange.dMaxFrequencyInMHz = 960;
    pcRFPowerScanner->SetRfSweepFrequencySpan( cError, tSweepRange );
    CHECK_VICOMERROR( cError );

    //

```

```

60     // Before any result can be obtained, the measurement has to be started
        //
        cout << "Start Measurement\n";
        pcRFPowerScanner->GetBasicInterface().StartMeasurement( cError );
        CHECK_VICOMERROR( cError );
65
        //
        // Now concrete measurement values are requested. The result shall give
        // the data prepared for a 124 line display (which reflects the
        // number of channels in E-GSM channel).
70     //
        CViComRFPowerScanInterfaceData::SSpecificParameters tRequestParams;
        tRequestParams.wRequestIdentifier = 0;
        tRequestParams.dwCountOfRequests = 5;
        tRequestParams.dwTimeBetweenRequestsInMs = 500; // half a second
75
        tRequestParams.eFreqPostProcess =

        CViComRFPowerScanInterfaceData::SSpecificParameters::etFreqPostProcess::SPECT
RUM_DISPLAY;
80     tRequestParams.FreqPostProcess.SpectrumDisplay.dMinFrequencyInMHz =
        tSweepRange.dMinFrequencyInMHz;
        tRequestParams.FreqPostProcess.SpectrumDisplay.dMaxFrequencyInMHz =
        tSweepRange.dMaxFrequencyInMHz;
85     tRequestParams.FreqPostProcess.SpectrumDisplay.dwCountOfDisplayLines = 124;
        tRequestParams.FreqPostProcess.SpectrumDisplay.eFreqDetector =

        CViComRFPowerScanInterfaceData::SSpecificParameters::UFreqPostProcess::SSpect
rumDisplay::FD_PEAK;
90
        tRequestParams.eTimeDetector =
        CViComRFPowerScanInterfaceData::SSpecificParameters::TD_RMS;

        //
        // The marker tool is enabled, so the result will show the strongest
95     // frequency available.
        //
        tRequestParams.bUseRequestForMarker = true;

        //
100    // Result power values shall be referenced to -140 dBm as basis, with a
        // resolution of 0.5 dB. Note that the theoretical maximum of the result
        // is fMinPowerValueInDBm + wMaxPowerInteger / fPowerResolutionInDB, in this
        // case -40 dBm.
        //
105    tRequestParams.PowerValueFormatSpec.fMinPowerValueInDBm = -140;
        tRequestParams.PowerValueFormatSpec.wMaxPowerInteger = 200;
        tRequestParams.PowerValueFormatSpec.ePowerWithOverflowFormat =

        CViComRFPowerScanInterfaceData::SSpecificParameters::SPowerValueFormatSpec::POWE
110    R_ONLY;
        tRequestParams.PowerValueFormatSpec.fPowerResolutionInDB = 0.5f;

        //
        // Now the request configuration is send to the interface. The 5 requests
115    // specified above are now issued in 5 second intervals, except the first
        // one, which should be calculated as soon as at least one raw measurement
        // is available.
        //
        cout << "Start Request...";
120    pcRFPowerScanner->RequestDerivedResult( cError, tRequestParams );
        CHECK_VICOMERROR( cError );
        cout << "ok\n";

        while ( true )
125    {
            //
            // Wait until at least two requests could be processed.
            //
            CViComRFPowerScanInterfaceData::SMeasResult* ptResult =
130            pcRFPowerScanner->GetResult( cError,

```

```

        2 * tRequestParams.dwTimeBetweenRequestsInMs );

    if ( ! ptResult )
        break;
135     cout << "Result " << " @ " << ptResult->dwPcTimeStampInMs << endl;
        cout << " # Samples: " << ptResult->dwCountOfPowerWithOverflowValues <<
endl;

140     float freq = 0.0;
        for ( int j=0; j<ptResult->dwCountOfPowerWithOverflowValues; j++ )
        {
            // Convert to MHz
            freq = ptResult->pMarkerResult->dMarkerFrequencyInHz / ( 1000 * 1000 );
145         double power = TO_DBM( ptResult->pwPowerWithOverflowValues[j] );

            cout << "\tValue " << j << " for Freq: " << freq << " MHz is " <<
150             power << "\t" << ptResult->pwPowerWithOverflowValues[j] << endl;

            if ( tRequestParams.bUseRequestForMarker && ptResult->pMarkerResult )
            {
155                 cout << "Marker: Freq: " << (freq)
                    << " MHz; Max: " << TO_DBM( ptResult->pMarkerResult->wMaxResultValue
)
                    << "; Min: " << TO_DBM( ptResult->pMarkerResult->wMinResultValue )
160                 << endl;
            }

            cout << "Shutdown\n";

165         //
            // Stop measurement and close the connection.
            //
            pcRFPowerScanner->GetBasicInterface().StopMeasurement( cError );
            CHECK_VICOMERROR( cError );

170         pcRFPowerScanner ->GetBasicInterface().HasMeasurementStopped( cError, 1000 );
            CHECK_VICOMERROR( cError );

            crRFPowerScanIfLoader.ReleaseTsmx( cError );
            CHECK_VICOMERROR( cError );
175     }

    /*****

180     int _tmain(int argc, _TCHAR* argv[])
        {
            try
            {
                cout << "ViCom RF Power Scan Demo" << endl;

185                RunViComRfPowerScanSample();
            }
            catch ( CViComError& cError )
            {
                cout << cError.GetErrorString() << endl;
190                return cError.GetErrorCode();
            }

            return 0;
        }

```

The code sample is organized in three sections: Global initialization is done from line 1 to line 21. The main part of the sample is then done in the function `RunViComRfPowerScanSample()`, starting at line 25. This method lasts until line 163 and is followed by the simple main function. The aforementioned function is discussed below.

Before any other work can be done, the TSMx must be initialized. This is done in line 35. A CViComLoader object takes care of the dll dynamic loading mechanism. The power scan interface object is configured to restrict the sweep range to the E-GSM band in lines 47 to 55. After that initialization, the TSMx is set into measurement mode in line 61.

In the lines 69 to 105 a measurement result request is configured. Therefore, the post process is first specified by telling the frequency detector what to do and by enabling the marker tool. Second, the output format is set. In line 114, the request is finally sent to the TSMx.

The results are evaluated in the code block in lines 118 to 152. Since it is not guaranteed to get a post processed result for all requests, this is done in an endless loop that quits when there is no result available after two requests should have been issued. If a result is available, it is formatted and printed to the console.

Since the marker tool has been enabled, there are two results available: The raw measurement samples for the marked frequency and the minimum/maximum calculation. Both results are displayed.

After the loop the application is shutdown and therefore stops the TSMx and unloads all allocated resources in lines 158-162.

In the screenshot in Figure 62 the output of one of the application runs is depicted. All of the five requests led to a result. The first two requests were each based on the data of one sweep. The third was made up of 8 raw measurements.

One thing that can be seen is that there is one single strongest transmitter on 949.926 MHz, although there is no good signal available. This is a GSM transmitter on channel 74.

```

c:\Programme\RuS ViCom\Application\ViComRFPowerScanSampleDebug.exe
ViCom RF Power Scan Demo
Loading TSMU...ok
Start Measurement
Start Request...ok
Result @ 25323579
# Samples: 1
Value 0 for Freq: 949.926 MHz is -89.5 101
Marker: Freq: 949.926 MHz; Max: -89.5; Min: -140
Result @ 25323592
# Samples: 1
Value 0 for Freq: 949.926 MHz is -91 98
Marker: Freq: 949.926 MHz; Max: -91; Min: -140
Result @ 25324399
# Samples: 8
Value 0 for Freq: 949.926 MHz is -92 96
Value 1 for Freq: 949.926 MHz is -91.5 97
Value 2 for Freq: 949.926 MHz is -92 96
Value 3 for Freq: 949.926 MHz is -90 100
Value 4 for Freq: 949.926 MHz is -90 100
Value 5 for Freq: 949.926 MHz is -89 102
Value 6 for Freq: 949.926 MHz is -92.5 95
Value 7 for Freq: 949.926 MHz is -91.5 97
Marker: Freq: 949.926 MHz; Max: -91; Min: -140
Result @ 25324779
# Samples: 4
Value 0 for Freq: 949.926 MHz is -91 98
Value 1 for Freq: 949.926 MHz is -90 100
Value 2 for Freq: 949.926 MHz is -93 94
Value 3 for Freq: 949.926 MHz is -89.5 101
Marker: Freq: 949.926 MHz; Max: -90.5; Min: -140
Result @ 25325380
# Samples: 6
Value 0 for Freq: 949.926 MHz is -94 92
Value 1 for Freq: 949.926 MHz is -90.5 99
Value 2 for Freq: 949.926 MHz is -93.5 93
Value 3 for Freq: 949.926 MHz is -92.5 95
Value 4 for Freq: 949.926 MHz is -89 102
Value 5 for Freq: 949.926 MHz is -91.5 97
Marker: Freq: 949.926 MHz; Max: -90.5; Min: -140
Shutdown
Press any key to continue...

```

Figure 62 Output of the sample application

10.5 RF Power Scan with TSMW

This chapter describes the special features of RF power scans using the TSMW. There are similarities with the power scan on TSMx devices, therefore the understanding of the measurement concept and the post processing chain of the previous chapter is required.

The first section describes the principal conceptual differences when using the TSMW compared to the TSMx implementation.

The second section contains a description of the interface structure for the power scans with TSMW. This section is important for the successful integration of the interface functionalities into a control application.

The sample application in the third section demonstrates the features of the interface and provides a reference for own developments.

10.5.1 Measurement and Post-Processing Concept

Similar to the TSMx version the basic measurement result of a power scan is a sweep over a defined frequency range. With the TSMW the sweep is done by acquiring data in time domain and calculating the spectrum using an FFT. Therefore additional parameters like the FFT size (→ frequency resolution) or window type (→ selectivity) are available to define the measurement. If the desired sweep range exceeds the maximum real time bandwidth of the TSMW, the frequency band is segmented. Normally the width of the segment is selected for maximum measurement speed, but the user has the possibility to overrule the automatic settings and define her own segment width.

After gathering the information in spectral domain the postprocessing follows the same concept as the TSMx version:

- Frequency detector to limit the number of spectral lines
- Time detector for averaging or peak detection in time domain
- Channel filter to perform sweeps over several subbands with fixed frequency raster and defined filter characteristics

Only one out of the three analysis modes in frequency domain – frequency detector, channel filter or raw data analysis – can be selected. With raw data analysis the measurement data is delivered right after the FFT without any further postprocessing. When the marker is enabled, no spectrum information but the frequency with the maximum power is returned. Depending on the settings of the time detector one or more power values are available. The marker is active for all three analysis modes.

10.5.2 Interface Concept

The ViCom interface for RF power scan with TSMW controls all parameters of a single sweep. Multiple instances of interfaces can run independently in a single application. Therefore it is possible to perform spectrum analysis over different frequency bands within a single control program; the same task can be done by using a channel filter in a single instance of the interface.

When starting a measurement the TSMW load has to be allocated. This is the only limiting factor for the number of interfaces running in parallel.

The interface is created and used with the following steps:

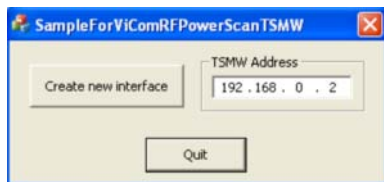
- Create a pointer to an instance of the interface by using the CViComLoader_TSMW function ConnectTSMW and identifying the TSMW by his IP address
- Prepare the sweep settings by filling the structure SSettings
- Transfer the settings to the power scan module with the interface function SetSettings
- Start the measurement with the interface function StartMeasurement
- Check the availability of measurement results with the interface function GetResultCounters
- Retrieve the measurement results with the interface function GetResult. The structure SResult holds all measured data
- Stop the measurement with the interface function StopMeasurement
- Release the interface using the CViComLoader_TSMW-function DisconnectTSMW

There are additional interface functions like GetSettings to retrieve the current sweep settings from the interface or setResultBufferDepth to define the number of results stored in the interface.

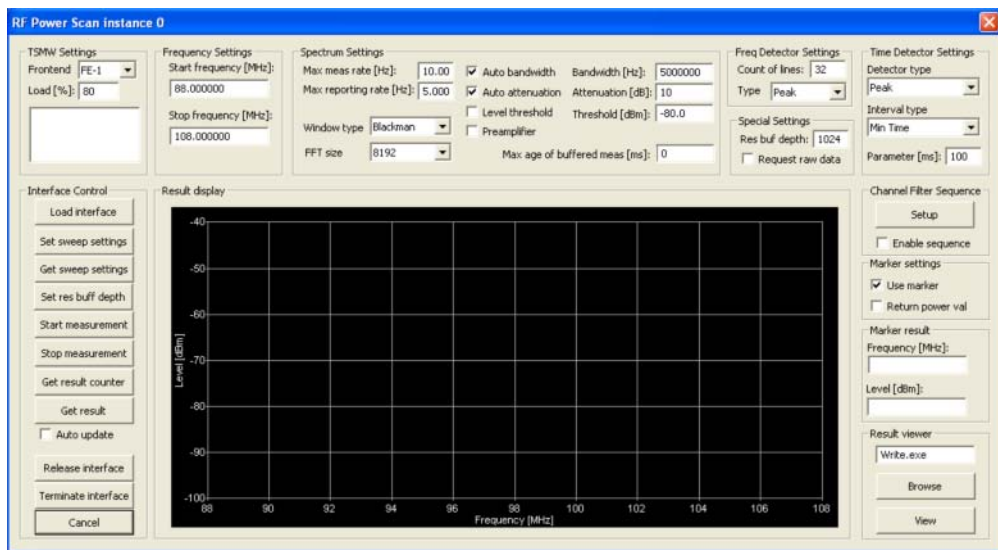
10.5.3 Sample application

The sample application is a simple program to demonstrate the capabilities of the RF power scan and to give examples for the interface intergration.

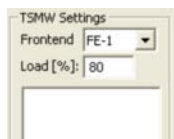
The sample application itself is only a launcher of interface instances. With each click on "Create new interface" a new instance of the interface is created which is controlled by an individual window.



The control window provides access to all interface functions, settings and results:



In the field “TSMW settings” the parameters of the hardware configuration and the resource allocation are defined. The display field shows simple status messages

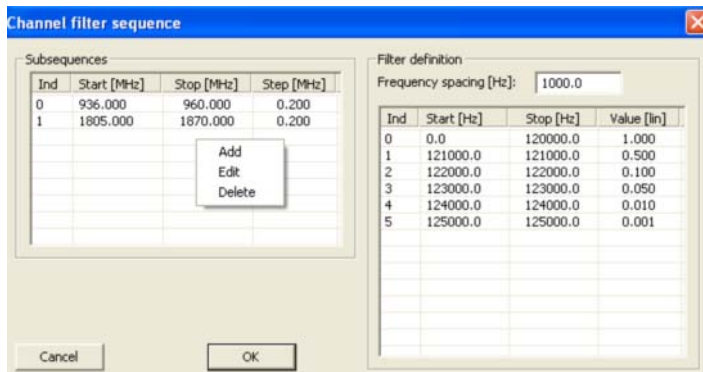


The channel filter mode is activated by checking the corresponding box. There is a special dialog window for the filter setup.



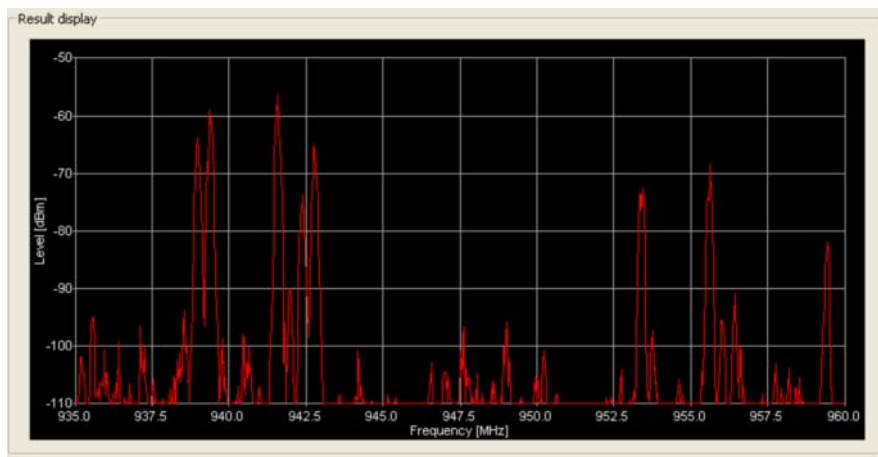
The channel filter setup consists of two parts:

- The list of subsequences defines the frequency ranges that will be analyzed.
- The passband shape of the filter is defined for individual points or ranges beginning at 0 (→ center frequency). The filter is symmetrical to the center. All frequency entries have to match the common frequency spacing.

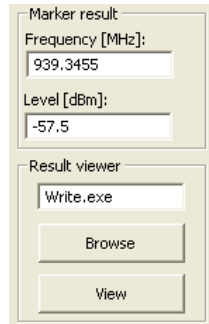


A right mouse button click on a list opens a menu to edit or delete existing entries or to add a new line.

The measured spectrum is displayed in a simple graph with auto-scaling. There is no listing of individual spectral levels. If the “Auto update” box is checked the results will be collected and displayed automatically, else a new result will be shown when the “Get result” button is clicked.



If the marker is enabled there is no spectrum data to display. Instead the frequency and the maximum power level of the marker is shown. In this case the individual results are collected in the Log file which can be viewed by clicking the “View” button.



10.6 RF Power Scanner Specific Trouble Shooting

This section lists some of the most common pitfalls when using the RF Power Scanner.

? *I try to get results from RF Power Scanner, but no matter how many lines I request, I only get a small set of results. What is wrong?*

! Maybe you have not set the `bUseRequestForMarker` variable explicitly of the interface class. In some situations, this may be set to true and therefore you won't get the number of results configured in `dwCountOfLinesRequested` or the number of channels.

? *I enabled the marker tool on my channel filtered data. The marked frequency is always in the first group of channels that I configured. What's wrong here?*

! This is a bug in current version and will be fixed in one of the next releases. If there are several subgroups configured in the channel filter that are not equidistant, the marker tool will only work on the first group.

? *I configured the frequency detector to return 124 lines, but I get sometimes multiples like 248 lines. How can I calculate which value belongs to which frequency, then?*

! This case might occur if the time detector is configured to return all samples with `CViComRFPowerScanInterfaceData::SSpecificParameters::eTimeDetector == TD_ALL`. In that situation, there might be data from more than one sweep returned, since no time aggregation is done. The result therefore consists of sequences of the output of the frequency detector. So, the frequency is of the n^{th} entry in the result is

$$\text{freq}(n) = \text{dMinFrequencyInMHz} + [n \% \text{dwCountOfDisplayLines}] * (\text{dMaxFrequencyInMHz} - \text{dMinFrequencyInMHz})$$

11 CW Measurements

In this chapter, the receiver channel power measurement module is described. Channel power measurement is similar to the inband power measurement available in the GSM network scanner, but it can be configured in a more flexible way using the CW measurements API.

Like in the previous chapters, this one begins with the description of the basic processing and calculation issues that arise in the different measurement modes. It is important to understand these basics to make the best use of the capabilities of the device.

The two subsequent chapters cover the sample application deployed with the ViCom shipment, and a simple programming example that shows how to use the API in your own code.

The TSML device that is capable of performing such measurements is called TSML-CW. In the subsequent sections, CW module and Receiver Channel Power Measurements API are used to describe the same type of technology and programming interface.



Please note: You can use the CW interface of ViCom on one device exclusively only. No other ViCom interfaces must be loaded on that receiver. The system will behave in unexpected ways (measurements will fail, for example) if this is done.

11.1 Channel Power Measurements

Up to a certain degree, all the measurements provided by the TSMx scanners are performed in the same way. The measured frequency is set in the receiver unit together with the bandwidth and the measurement is started after some pause time. Incoming data is converted to digital signals, demodulated and finally reduced to a downsampled set of IQ data.

Depending on the measurement mode, 512 IQ data pairs (so called raw measurement data) are written into an internal memory buffer and an interrupt is raised after some time elapsed. Additionally, an externally connected trigger device generates pulses that are also recorded and counted. The interrupt handling routine then transfers some of the raw data chunks to an internal ring buffer. Depending on the trigger mode, all chunks or only a subset of them are stored in that ring buffer.

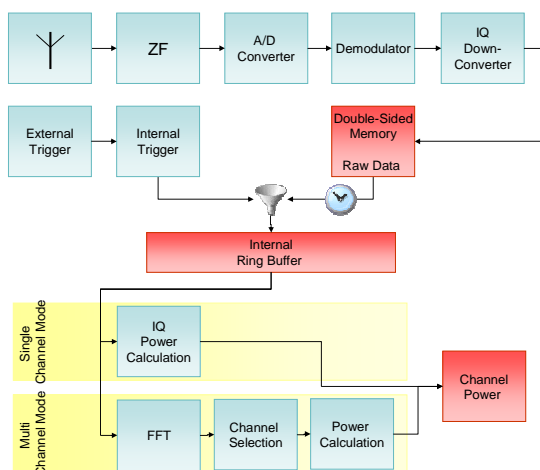


Figure 63 Channel Power Measurement

From the external triggers, the number of times they have been recorded and the time intervals between those triggers, internal triggers are generated. These internal triggers are the basic element to decide which raw measurement blocks are moved to the internal ring buffer, which in turn is used as input for different calculation routines. For more details on the triggering, refer to the section about triggering below.

11.1.1 Measurement Modes

These two different calculation routines in the TSMx are called Single Channel and Multi Channel measurement modes. Both differ in the accuracy, sensitivity to noise and measurement speed they provide, where the single channel measurement is used to focus the measurement on one specific channel, being as accurate as possible and reducing the influence from strong senders in the neighbourhood. A faster way to do the measuring is provided by the multi channel measurement mode, which in turn leads to stronger noise levels.

As input both modes require a list of channels to be defined. Each channel consists of a center frequency, a bandwidth and a measurement time specification. The measurement mode can be specified for each single channel, the TSMx software will then combine related channels and measurement modes appropriately to make the best use of the available processing time.

11.1.1.1 Single Channel Measurement

The concept of the single channel measurement mode is very simple. The internal hardware of the TSMx is set to a specific frequency and a specific bandwidth, which must not exceed 4 MHz. Then the hardware measurement is performed for a specific amount of time. The results of this data lead to a whole bunch of IQ-samples, which in turn are then used to calculate the channel power. The overall result is the sum of the squares of the real and imaginary part of the IQ samples.

This direct approach leads to very accurate results, since the attenuator is set to best-fit the measured channel and the IQ samples can be used directly and completely to calculate the power value. All this has to be paid off by blocking the measurement unit for the measurement time where only one channel is measured, even if the bandwidth is significantly smaller than the maximum. In that way, all channels have to be measured sequentially, so that one sweep will last at least the sum of the measurement times (there are additional costs, because the equipment must be synchronized internally).

All this is depicted in the figure shown below. Each of the three channels configured is measured on its own. The read bar in the mid of the channels is the center frequency and the blue background shall indicate the bandwidth.

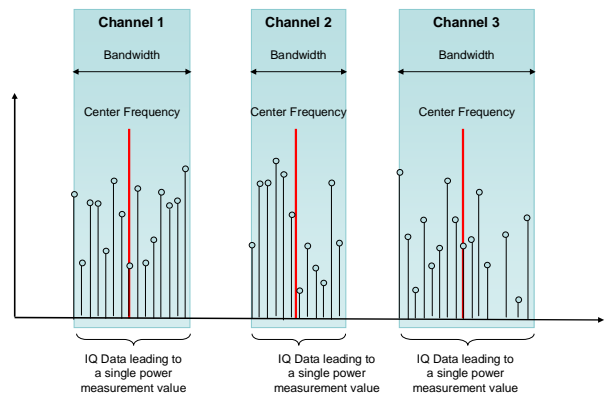


Figure 64 Single Channel Measurement

11.1.1.2 Multi Channel Measurement

To gain high measurement performance the channels specified to be measured are grouped together to cover a bandwidth of at most 4 MHz in each group. One such group is then measured together, which leads to higher measurement rates since the frequency has to be set only once in the hardware.

There are several drawbacks of this measurement method. One problem that might arise is that the attenuator is set for the set of frequencies, which may lead to higher noise levels in the case of some strong channels and the others being relatively weak.

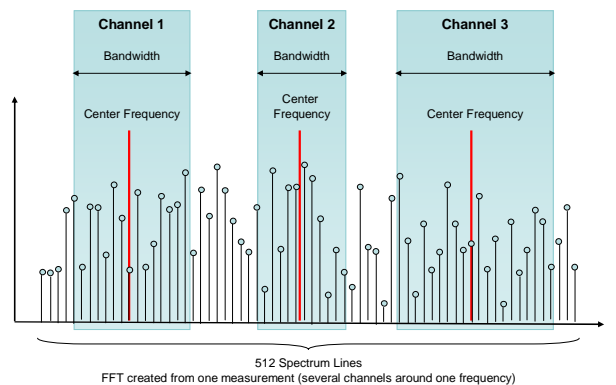


Figure 65 Multi Channel Measurement

Grouping Algorithm

The grouping mechanism is a sophisticated algorithm to also assure that at least 10 spectrum lines in the resulting spectrum of the FFT contribute to the overall power value for one channel.

For example, if two channels shall be measured with a bandwidth of 50 kHz at frequencies 400.1 and 400.15 MHz, and one with 3.1 MHz at 401.7 MHz. This would mean that the total bandwidth is 3.2 MHz, and since this bandwidth would be split into 512 spectrum lines, the distance between each line is 6.25 kHz. In this case two groups would be created from these three channels, because the 6.25 resolution would not lead to 10 power values for the channels with 50 kHz bandwidth.

11.1.2 Measurement Scheduling

One major difference to the other APIs in the ViCom library is that the measurement results are not necessarily delivered when the internal processing is finished or some time has elapsed. One of the available operation modes works in that way, but there is also the possibility to make the result creation depending on an external trigger device.

11.1.2.1 Free Run

The Free Run Mode works without external triggering and processes the raw data as soon as enough chunks are available to serve the required measurement time. In single channel mode, that means that one channel is measured until the total time required to measure the raw chunks exceeds the required measurement time. Based on these chunks, the overall power is calculated as described in the following chapter about power value aggregation.

In multi channel mode, the raw measurement chunks are processed by a FFT, and the spectrum lines belonging to one channel of the channel group are used to calculate the single power values.



Different measurement times in a frequency cannot be used in this mode. The maximum measurement time is used in this case to record the raw measurement data, and for all channels the data of all resulting FFTs is used.

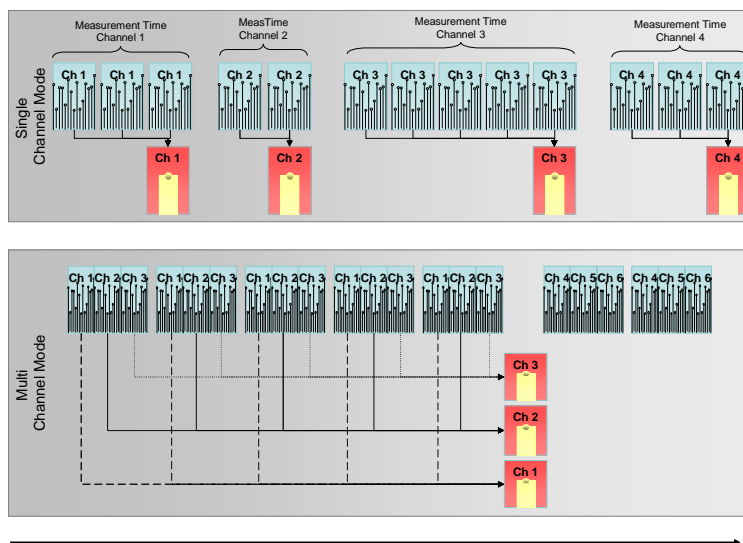


Figure 66 Free Run Mode

11.1.2.2 Time and Distance Triggering

To reduce the amount of data used to calculate channel power results, external trigger devices can be connected to the TSMx device. Each time a trigger is fired, the event is logged with its timestamp. As soon as a raw measurement chunk is available, the TSMx software checks if the chunk belongs to a certain trigger, and if so, the chunk (and possible the following ones as well) will be used to calculate the channel power.

When triggering is active, the trigger signal from the external device can be filtered to generate internal triggers that result in the creation of a measurement result. For example, if it is useful to use only every 10th trigger, this can be configured. It is even possible to specify non integers as trigger filter, as long as the rational does not get less than 1. As a consequence, it is not possible to create more internal trigger signals by software than actually fired by hardware. In the subsequent sections, the trigger events from the trigger device are called external triggers, and the calculated ones are called internal triggers.

As soon as the trigger factor is set to a value that is not equal to 1 (i.e. every external trigger is exactly the same as an internal trigger for the result creation), the CW module must extrapolate the timestamp when the output trigger has to be generated based on the history of the last input triggers.



This extrapolation step is of course not a 100% exact replacement and the triggers raised in that way might differ from the ones really measured, especially when integer multiples are used. For example, if the factor is set to 2 it is not necessarily true that each second input trigger leads to an output trigger, since the calculation based on the time intervals of the trigger history might be different (see figure below).

In the figure below, several examples are shown how the internal triggers are derived from the external triggers. In the simplest case, the external triggers exactly match the internal ones, which is the case when 1:1 triggering is active.

If a 3:2 triggering is active, the first two external triggers are used to extrapolate the timestamp of the internal trigger. This step is repeated when the next external triggers arrive. In the sample, the distance between the first and second internal trigger is increased because the third external trigger also is raised after a longer pause. The fourth external trigger is fired faster and therefore the internal trigger also is fired nearly immediately.

In the last case, a 2:1 triggering is shown. The time delta between two subsequent external triggers is therefore doubled to calculate the internal trigger timestamp (although the real algorithm is somewhat more complicated). An interesting detail is that the fourth external trigger is ignored, since the internal trigger calculated from external trigger two and three is after the one calculated from the third and fourth.

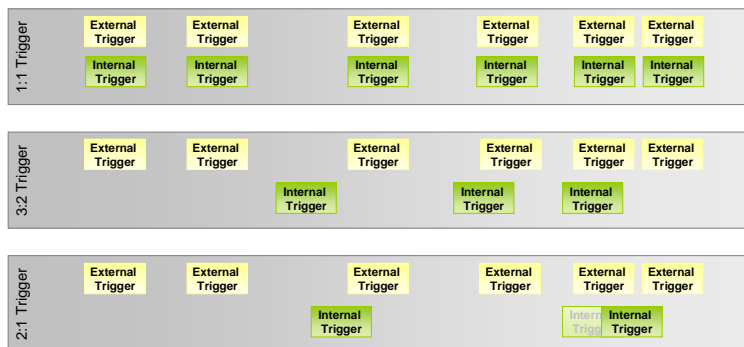


Figure 67 – Internal Trigger Calculation

Until the first internal trigger for a sweep is raised, the raw measurement data is ignored and not stored for later power value calculation. The chunk that is associated with the time interval that contains the extrapolated timestamp for the internal trigger, is the first used to calculate the power value for the channel. Depending on the measurement time specified for one channel, the additional chunks have to be taken into account to calculate that value.

In single channel measurement mode this means that also parts of a chunk can be used to calculate the power value. As depicted the figure below, the internal trigger timestamp is in located in the mid of the time interval associated with the third raw measurement chunk. Therefore, the second half of that chunk and the first of the next are used to calculate a raw power value. In total, three raw power values are calculated from four chunks. From these three raw power values, the overall channel power is calculated.

Once all chunks required to fulfill the measurement time constraint for the channel are available, the measurement unit is tuned to the next frequency and the raw measurements are processed immediately in the same way as before.



It is important to note that there is not additional trigger required, since once the internal trigger is located in a chunk, all configured channels are then measured. As a consequence, the internal trigger intervals must not be smaller than the total sweep time required to setup, measure and process all channels.

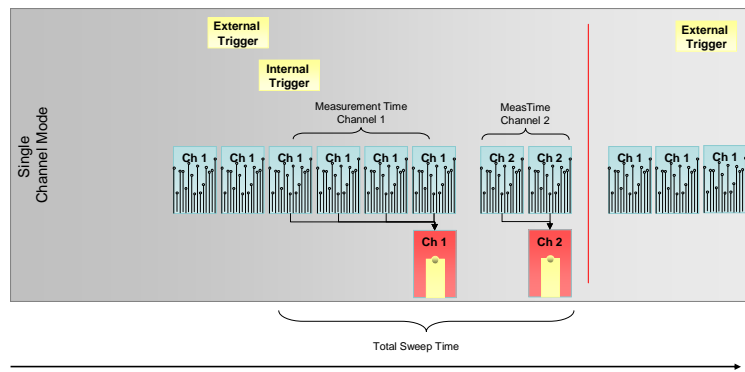


Figure 68 – Triggering in Single Channel Mode

The multi-channel mode basically works in the same way. The raw measurement data is only processed with the FFT and the spectrum is analysed after the internal trigger is raised. In comparison to the single channel mode, the raw data is not split to only use parts of a chunk or to combine to chunks for a FFT. Chunks are always used completely or ignored in this mode.

In the sample shown below, the tuning to the channel group containing the channel 4, 5 and 6 occurs after all channels in the first group have been measured once.

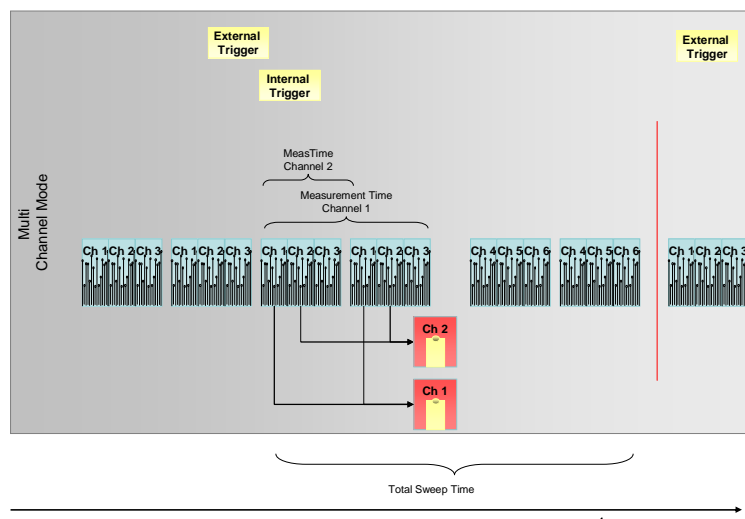


Figure 69 Triggering in Multi Channel mode

One important difference between the Single and Multi Channel Measurement mode is illustrated in the figure above as well. In case of the Multi Channel Mode, the measurement time of one set of frequency that is measured in a row is chosen to best-fit the maximum measurement time of the included channels. For example, if the measurement times for channels 1, 2 and 3 are 300, 200 and 500 μ s, then the data used to calculate the channel powers for those channels is taken from raw measurements received within about 500 μ s.

11.1.3 Channel Power Aggregation

In case that multiple channel power values have to be merged to create one final result (because more than one raw measurement chunk shall be used to calculate the overall power for a channel/channel group), a strategy must be chosen how this can be done. From the raw measurement data, i.e. either the subset of spectrum lines from the FFT in case that the Multi Channel measurement mode was active, or the IQ samples from the IQ down-converter in case of the Single Channel measurement mode, power values are calculated for each raw power value group.

If more than one group of 512 power values is within the measurement time for a channel (in the figure depicted below, there are four such chunks from which four power values are created), then from the set of these values one result has to be calculated for the channel for which the measurement was configured.

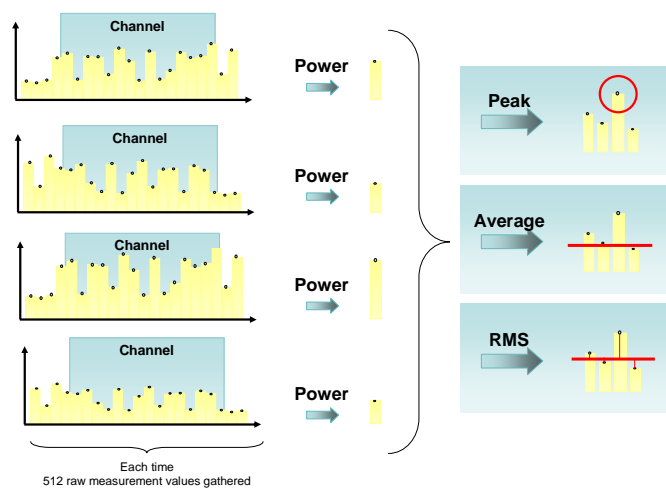


Figure 70 Power Aggregation Modes

The aggregation mode can be chosen from one of the following three options:

- **PEAK:** Select the maximum power value from the single measurement results.
- **AVERAGE:** Calculate the linearized average of all the power values calculated for the different raw measurements, where p is the power value at of the raw measurement i , and N is the number of measurements contributing to the overall result.

$$p_{avg} = \frac{1}{N} \sum_{i=1}^N p_{f_i}$$

- **ROOT-MEAN-SQUARE:** The root mean square of all intermediate results is calculated based on the equation shown below. As for the average, the power values are converted to Watt to have a linear scale for calculating the final result.

$$p_{rms} = \sqrt{\sum_{i=1}^N p_{f_i}^2}$$



The last two modes do not use the calculated power values directly, but first linearize the power values. Otherwise the aggregation would lead to uncorrect results.

11.1.4 Sample Application

In the same way as shown in the other ViCom APIs, a sample application is packaged with the RF Channel Power measurements module. It has a similar look and feel as the other sample applications and shares the same purpose: The usage of the API shall be shown as a GUI in the same way as it looks in the code, to reduce the time to get familiar with the API and to make trial and error possible without coding.

Basically, the user interface looks familiar if you have used one of the other sample applications deployed with the other modules of the ViCom library. The application shows one big dialog that contains three columns of functionality: The left column holds the buttons that correspond to the functions the interface provides. The structures given to these functions are filled with the content of the controls shown in the central column (and partially with some of the right one).

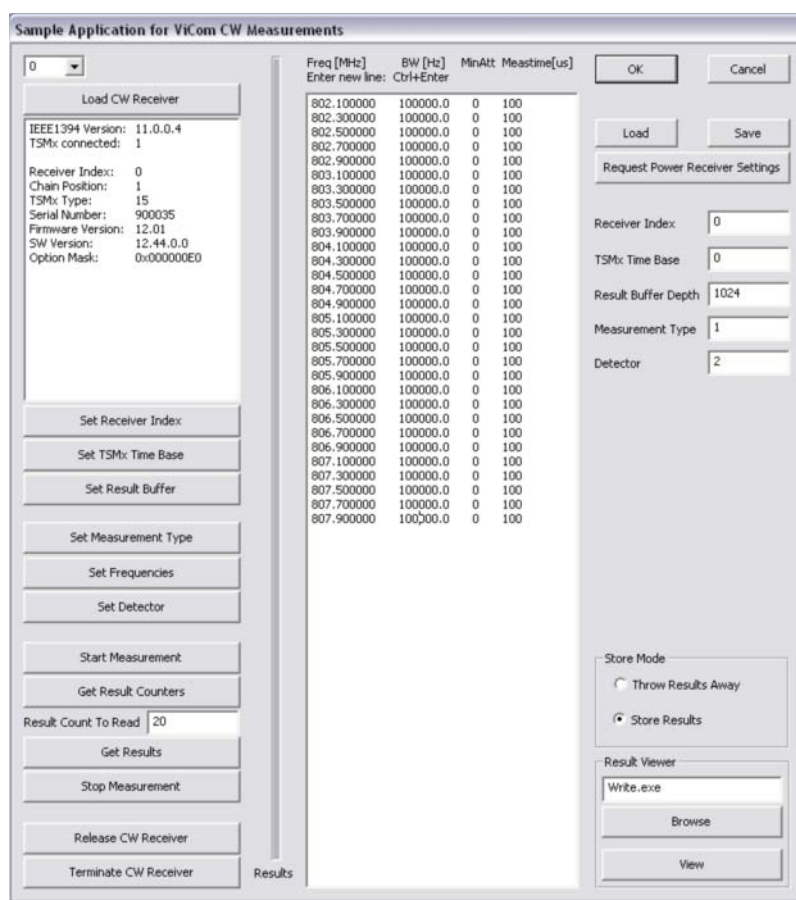


Figure 71 Sample application for ViCom CW

In this demo application, the usage is straight-forward. The most important settings can be made in the central column, where the measured channels are configured, and in the right column in the text boxes “Measurement Type” and “Detector”.

In this central column, the list of frequencies that shall be measured can be specified. Each line in the edit field must contain one channel specification, consisting of the measured frequency in MHz, the bandwidth in Hz, the attenuation mode and the measurement time in microseconds.



It is important to specify all of these settings. If there are less than four columns entered (separated by at least one space character), the line is rejected when clicking “Set Frequencies” button.

New lines can be entered by pressing Ctrl+Enter. Once the frequencies are specified, a click on “Set Frequencies” transfers these settings to the TSMx, if all validation routines have been passed successfully.

The measurement type can be one of the values listed below, and is set using the “Set Measurement Mode” button.

- 0 – Single Channel Mode, each channel is measured until one power value can be calculated (based on the measurement time specified)
- 1 – Multi Channel Mode, Channels are grouped and measured together if possible

As detector, the following values are valid, which can be set by pressing “Set Detector”.

- 0 – Peak detection (highest power value is chosen when multiple raw measurements are available)
- 1 – The raw data is averaged (i.e. the linearized power values are averaged)
- 2 – Root-Mean-Square: The single (linearized) values from the raw measurements are squared, and the square-root of the sum of the power values makes the final power value.

Once configuration is finished, the measurement can be started by clicking on “Start Measurement”. After checking the buffer state with “Get Result Counters”, the number of results specified in “Result Count To Read” can be removed from the buffer by pressing “Get Results”.

If the “Store Results” radio button is active when fetching the results, they are written into the ViComMeasurements.txt file stored in the Application\LogFiles directory. The content of the file looks similar to the one shown in the figure below.

Each result contains a number of sweeps, where the frequencies are given in ascending order. Note that the sweeps are not necessarily delivered completely, parts of sweeps maybe delivered with the next result, as shown in the figure.

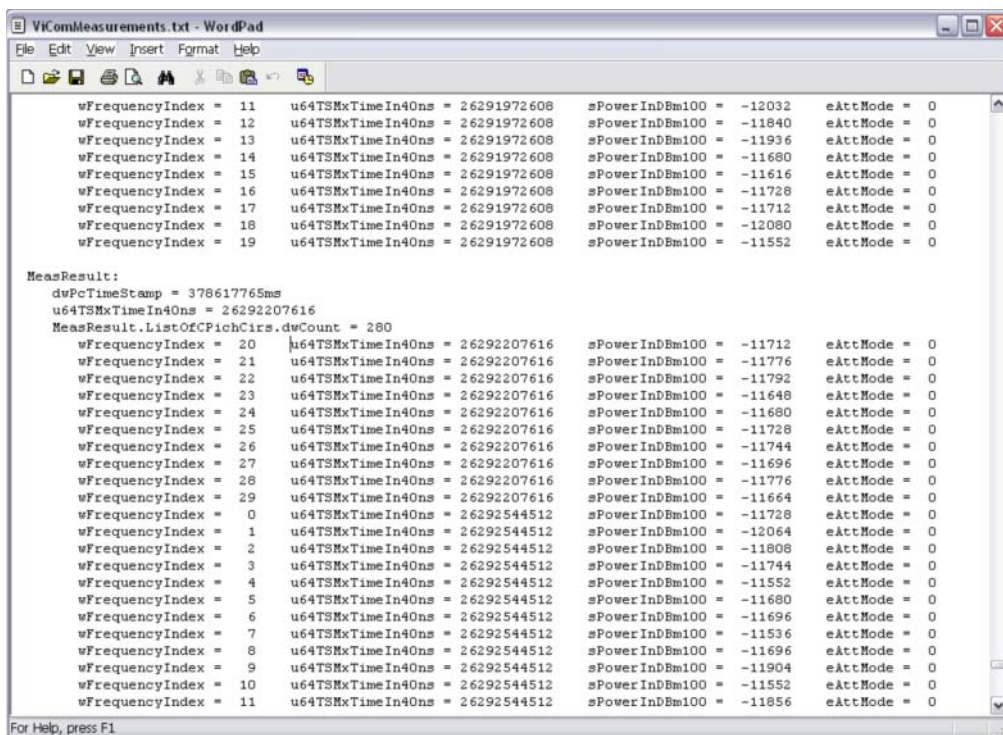


Figure 72 Screenshot of Measurement Result

11.2 Code Example

To demonstrate the usage of both measurement modes, the code sample below first performs some measurements in the multi-channel mode to find the three strongest channels in 100 measurements. These channels are then measured in single channel mode, but power values are only reported if the value is higher than the third strongest peak measured in the first take.

In order to make the single channel measurement do its job, the previously started multi-channel measurement has to be stopped, and the measurement task has to be reconfigured.

```
#include "stdafx.h"

#include <windows.h>
#include <iostream>
5 #include <algorithm>
#include <conio.h>

#include "ViComBasicInterface.h"
#include "ViComError.h"
10 #include "ViComLoader.h"
#include "ViComCWErrors.h"
#include "ViComCWInterface.h"
#include "ViComCWInterfaceData.h"

15 using namespace std;

/*****
//
//
```

```

20 // Helper macro to convert ViCom return codes and errors to exception to
// install a global error handling = exception catching at one place. Also
// supports a simplistic kind of progress by showing what action is currently
// performed.
//
25 #define EXEC_VICOM( msg, call ) \
    if ( strlen(msg) ) { cout << msg << "..."; } \
    call; \
    if ( cError.GetErrorCode() > 0 ) { \
30     char* szErr = new char[strlen(cError.GetErrorString()) + 1]; \
    strcpy(szErr,cError.GetErrorString()); \
    throw CViComError(cError.GetErrorCode(), szErr); } \
    if ( strlen(msg) ) { cout << "ok" << endl; }

//*****

35 //
// Helper class to make sorting possible and simplify detection of the
// strongest 3 senders. This struct supports sorting using sort() function
// of the stl.
40 //
struct SChannelPowerInfo
{
    int nChannelIndex;
    double fMaxChannelPower;
45
    bool operator < ( const SChannelPowerInfo& c )
    {
        //
        // We use the greater operator to make the ascending order that
50     // is created by sort() start with the strongest sender, and is
        // then ordered descending.
        //
        return fMaxChannelPower > c.fMaxChannelPower;
55 } ;

//
// This function is necessary to make sort() work on SFrequencySettings
// datastructure. The channels that are configured in a measurement task
60 // must be sorted in ascending order by their frequency, which is
// accomplished using this less than operator
//
bool operator < (
65     const CViComCWInterfaceData::SFrequencySetting& f1,
    const CViComCWInterfaceData::SFrequencySetting& f2 )
{
    return f1.dCenterFrequencyInMHz < f2.dCenterFrequencyInMHz;
}

70 //*****

void RunViComCWSample()
{
    CViComError cError;
75     CViComLoader< CViComCWInterface > cCwInterface;

    //
    // First thing is to load the application dll and to initialize the TSMx
    // device for communication.
80     //
    EXEC_VICOM( "Loading Continuous Wave Measurements API",
        cCwInterface.LoadTsmx( cError ) );

    EXEC_VICOM( "Retrieving special interface",
85     CViComCWInterface* pcCwInterface =
        cCwInterface.GetpInterface( cError ) );

    //
90     // Select the first TSMx in the FireWire chain to be used as measurement
    // device.

```

```

//
EXEC_VICOM( "Setting receiver index",
    pcCwInterface->GetBasicInterface().SelectReceiver( cError, 0 ) );
95
//
// Store as many results in the internal buffer as possible, although the
//
// processing in this sample is performed immediately and the buffer should
never
100 // be filled completely.
//
CViComBasicInterfaceData::SResultBufferDepth cResultBufferDepth;
cResultBufferDepth.dwValue =
105 CViComBasicInterfaceData::SResultBufferDepth::dwMax;

EXEC_VICOM( "Setting result buffer",
    pcCwInterface->GetBasicInterface().SetResultBufferDepth( cError,
    cResultBufferDepth ) );

110
//
// The aggregation algorithm specifies how the different samples gained over
time
// shall be aggregated. There are three different modes, of which the root-
mean-
115 // square calculation is chosen here. Should be set before the frequency
table is
// specified.
//
EXEC_VICOM( "Setting detector",
120     pcCwInterface->SetDetector( cError,
        CViComCWInterfaceData::etDetector::TD_RMS ) );

//
125 // In the first step, the 124 channels are measured in multi-channel mode.
// As channels the first 124 GSM 900 channels are used.
//
const int MULTI_CHANNEL_COUNT = 124;
CViComCWInterfaceData::SChannelSettings cChannelSettings;
130 cChannelSettings.dwCount = MULTI_CHANNEL_COUNT;

for ( int i=0; i<MULTI_CHANNEL_COUNT; i++ )
{
135     cChannelSettings.TableOfFrequencySetting[i].dBandwidthInHz = 200 * 1000;
// == 200 KHz
cChannelSettings.TableOfFrequencySetting[i].dCenterFrequencyInMHz
    = 935.1 + i * 0.2; // specify how long each channel shall be measured
cChannelSettings.TableOfFrequencySetting[i].dwMeasTimeIn_us = 100;
// attenuation is turned off
140 cChannelSettings.TableOfFrequencySetting[i].eMinAttenuation =
    CViComBasicInterfaceData::STSMxAttenuation::TSMX_ATT_PA_OFF_RF0_IF0;
}

EXEC_VICOM( "Setting frequency table",
145     pcCwInterface->SetFrequencyTable( cError, cChannelSettings ) );

EXEC_VICOM( "Setting measurement type to multi channel",
    pcCwInterface->SetMeasurementType( cError,
150     CViComCWInterfaceData::MT_MultiChannel ) );

//
// All configuration is done, so the measurement can be started.
//
EXEC_VICOM( "Start measurement",
155     pcCwInterface->GetBasicInterface().StartMeasurement( cError ) );

//
160 of // The first measurement turn is used to find the strongest sender in the set
// scanned frequencies. To do this, the maximum value of all results for one

```

```

// channel is stored.
//
165 const DWORD MULTI_CHANNEL_MEASUREMENT_COUNT = 100;
    DWORD dwNumberOfMeasurements = 0;

    SChannelPowerInfo vMaxChannelPower[ MULTI_CHANNEL_COUNT ];
    for ( int i=0; i<MULTI_CHANNEL_COUNT; i++)
170 {
        vMaxChannelPower[i].nChannelIndex = i;
        vMaxChannelPower[i].fMaxChannelPower = -200;
    }

    CViComCWInterfaceData::SMeasResult* pcResult = NULL;
175 while ( ( pcResult = pcCwInterface->GetResult( cError, 5000 ) )
        && dwNumberOfMeasurements < MULTI_CHANNEL_MEASUREMENT_COUNT )
    {
        SViComList< CViComCWInterfaceData::SMeasResult::SPowerResult
180 >::SLinkedObject*
        pcPowerResult = pcResult->ListOfPowerResults.pFirst;
        for ( unsigned int i=0; i<pcResult->ListOfPowerResults.dwCount; i++ )
        {
            //
            // One result can contain an arbitrary number of (complete) sweep
185 results,
            // and each time the frequency index passes 0, one additional
            measurement
            // sweep is finished.
            //
190 if ( pcPowerResult->wFrequencyIndex == 0 )
                ++dwNumberOfMeasurements;

            cout << " Frequency:" << 935.1 + 0.2 * pcPowerResult->wFrequencyIndex
195 << " MHz:"
                << " Power: " << (float)pcPowerResult->sPowerInDBm100 / 100.0 << "
                dB"
                << " Att Mode: " << pcPowerResult->eAttMode << endl;

            vMaxChannelPower[pcPowerResult->wFrequencyIndex].fMaxChannelPower =
200 max(
                vMaxChannelPower[pcPowerResult->wFrequencyIndex].fMaxChannelPower,
                pcPowerResult->sPowerInDBm100 / 100.0 );
            pcPowerResult = pcPowerResult->pNext;
        }
205 }

    //
    // Now the measurement is stopped and the configuration is changed to measure
210 the
    // strongest frequency from the channels in single measurement mode.
    //
    EXEC_VICOM( "Stopping measurement",
                pcCwInterface->GetBasicInterface().StopMeasurement( cError ) );
    EXEC_VICOM( "Waiting for measurement to be stopped",
215 pcCwInterface ->GetBasicInterface().HasMeasurementStopped( cError, 5000 )
    );

    //
220 // Now its time to look for the channel with the highest power value and
    monitor
    // it in the single channel mode. The sorting operator used inverts the
    normal
    // order to be descending, which is simpler to process later (highest power
225 value
    // is the first entry etc.)
    //
    sort( vMaxChannelPower, vMaxChannelPower + MULTI_CHANNEL_COUNT );

230 const int SINGLE_CHANNEL_COUNT = 3;
    cChannelSettings.dwCount = SINGLE_CHANNEL_COUNT;

```



```

235 //
// These are basically the same settings as used in the multi channel
// measurement.
//
for ( int i=0; i<SINGLE_CHANNEL_COUNT; i++ )
{
240     cChannelSettings.TableOfFrequencySetting[i].dBandwidthInHz = 200 * 1000;
    cChannelSettings.TableOfFrequencySetting[i].dCenterFrequencyInMHz =

cChannelSettings.TableOfFrequencySetting[vMaxChannelPower[i].nChannelIndex]
    .dCenterFrequencyInMHz;
245     cChannelSettings.TableOfFrequencySetting[i].dwMeasTimeIn_us = 100;
    cChannelSettings.TableOfFrequencySetting[i].eMinAttenuation =
        CViComBasicInterfaceData::STSMxAttenuation::TSMX_ATT_PA_OFF_RF0_IF0;
}

250 //
// The list of frequencies must be ordered in ascending order. This is
// done by the line shown below. Otherwise the call to SetFrequencyTable()
// would result in an error.
//
255 sort( cChannelSettings.TableOfFrequencySetting,
        cChannelSettings.TableOfFrequencySetting + SINGLE_CHANNEL_COUNT );

EXEC_VICOM( "Setting frequency table",
            pcCwInterface->SetFrequencyTable( cError, cChannelSettings ) );

260 EXEC_VICOM( "Setting measurement type to single channel",
            pcCwInterface->SetMeasurementType( cError,
            CViComCWInterfaceData::MT_SingleChannel ) );

265 //
// Second configuration is done, so the measurement can be started again.
//
EXEC_VICOM( "Start measurement",
            pcCwInterface->GetBasicInterface().StartMeasurement( cError ) );

270 while ( ( pcResult = pcCwInterface->GetResult( cError, 10000 ) ) )
{
    //
    // This time the measurement is done until the user presses a
    // key.
275     //
    //
    if ( _kbhit() )
        break;

    SViComList< CViComCWInterfaceData::SMeasResult::SPowerResult
280 >::SLinkedObject*
    pcPowerResult = pcResult->ListOfPowerResults.pFirst;
    for ( unsigned int i=0; i<pcResult->ListOfPowerResults.dwCount; i++ )
    {
285         double fPower = (float)pcPowerResult->sPowerInDBm100 / 100.0;

        //
        // Only show power values that are better than the lowest of the three
        // top-values measured before.
        //
290         if ( fPower > vMaxChannelPower[2].fMaxChannelPower )
        {
            cout
                << " Frequency:"
295                 << cChannelSettings.TableOfFrequencySetting[
                    pcPowerResult->wFrequencyIndex ].dCenterFrequencyInMHz
                << " MHz, Power: "
                << (float)pcPowerResult->sPowerInDBm100 / 100.0
                << " dB, Att Mode: "
                << pcPowerResult->eAttMode << endl;
300         }

        pcPowerResult = pcPowerResult->pNext;
    }
}

```

```

305     }
        cout << "Shutdown\n";

        //
        // Stop measurement and close the connection.
310     //
        EXEC_VICOM( "Stopping measurement",
            pcCwInterface->GetBasicInterface().StopMeasurement( cError ) );

        EXEC_VICOM( "Waiting for measurement to be stopped",
315         pcCwInterface ->GetBasicInterface().HasMeasurementStopped( cError, 5000 )
    );

        EXEC_VICOM( "Cleaning up device",
320         cCwInterface.ReleaseTsmx( cError ) );
    }

    /*****/

325 int _tmain( int argc, _TCHAR* argv[] )
    {
        try
        {
            cout << "ViCom CW Scanner Demo" << endl << endl << endl;

330         RunViComCWSample();
        }
        catch ( CViComError cError )
        {
            cout << cError.GetErrorString() << endl;
335         delete [] cError.GetErrorString();
            return cError.GetErrorCode();
        }

        return 0;
340    }

```

This code sample is structured in the same way the other parts of this documentation. The last part includes the main program that simply performs the error handling in form of a try-catch block, which can be an appropriate kind of error handling since the error messages of the ViCom API are helpful for the programmer. In an end-user oriented application however they should be converted to be understandable for non technicians. Within the try-catch, the one and only function that contains all the logic is called (line 311).

In lines 41 to 68 some helper structures and functions for sorting are defined. They are used later in combination with the STL function `sort`, and at least the operator< implementation for the structure `CViComRFChannelPowerReceiverInterfaceData::SFrequencySetting` is helpful and can be reused in other applications (see below).

The first action when working with ViCom interface is always to load the required interface using the `CViComLoader<>` template class and call the `LoadTsmx()` method. It is then possible to receive the interface to control the specific functionality, calling `GetpInterface()`. If more than one device is connected, it is important to specify the receiver unit that shall be used for measuring. In the example, always the first device is used. All this is shown in lines 81 to 93.

In lines 122 to 139 the measured frequencies are set. These are GSM 900 downlink channels, starting at 935 MHz and ranging to 960 MHz, each channel having a bandwidth of 200 kHz. From this set of channels, the three strongest senders shall be detected. Each channel shall be measured for 100 μ s before a power value is calculated. As aggregation function, the Root-Mean-Square method is used, which is specified in lines 113-115.

This first measurement is performed in the multi-channel measurement mode, to make the measurement run as quickly as possible. This can be seen in lines 141 to 143. The internal grouping algorithm will then build 7 groups, each containing up to 4 MHz (20 channels).

After the measurement has been started (lines 148-149), 100 power channel values are evaluated to find the peak value for each channel. Because each result can contain a whole bunch of power values from multiple sweeps, the count of measurements evaluated is increased each time the frequency index is set to 0 again (lines 180,181).

The peak values are stored in a structure that also memorizes the index of the according channel (lines 188-191). Once all 100 measurements are evaluated, the measurement is stopped.

Using the STL sort() algorithm, these values are then sorted to find the strongest senders. This function uses the < operator of the arguments implicitly and orders the given set in an ascending order. Since greater values are required to be ordered before smaller ones, the implemented operator< for the SChannelPowerInfo structure uses the > operator to achieve a descending sorted array (line 212, 46-54).

Next a new measurement is set up in the same way for the multi channel measurement performed before, but this time the single channel measurement mode is used, because measurement performance is now not important, but higher accuracy is desired. The main difference is that the channel list is sorted by frequency in line 237 ff. since the ViCom requires the specified channels to have increasing frequencies (the internal, optimizing algorithms require this constraint to perform an efficient grouping).

After the second measurement is started, all the power values are reported that are higher than the smallest peak value of the three channels that are now observed in single channel mode (line 272 ff). The measurement is performed unless an arbitrary key is pressed again.

Finally, the measurement is stopped and the device resource is freed for other usage again. The output of the result is depicted below.

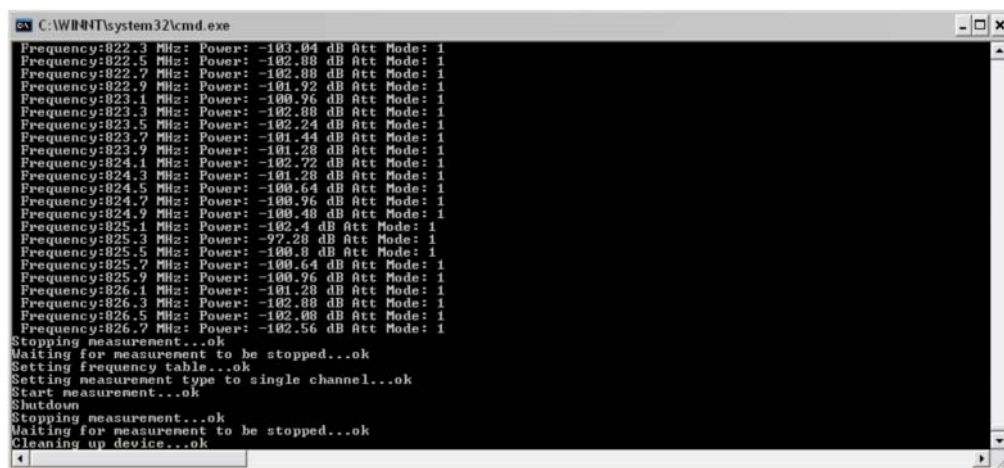


Figure 73 Sample output

11.3 Error Codes

The table below shows some of the possible error codes that are returned by the CW ViCom interface. This list is not complete, but contains the codes that are related to the most common pitfalls in the interface.

Error Code	Error Description
13004	VICEC_RF_CHANNEL_POWER_RECEIVER_INTERNAL_IBPM_INIT Returned when calling StartMeasurement() and the TSMx device has not the appropriate option installed to perform CW measurements.
13005	VICEC_RF_CHANNEL_POWER_RECEIVER_FREQ_OUT_OF_RANGE Returned when setting frequencies using SetFrequency() and the one of the frequencies in the list is less than 80 MHz or greater than 3 GHz (resp. 6 GHz if the TSM-CW device is used)
13006	VICEC_RF_CHANNEL_POWER_RECEIVER_FREQ_NOT_IN_ASCENDING_ORDER Frequencies must be sorted in ascending order when given to SetFrequencies(). This is necessary since the ordering is necessary to efficiently create frequency groups in Multi Channel measurement mode and an internal reordering would complicate the frequency index reporting in the result. See the code sample to find a way to simply sort the frequency list before the measurement is started.
13009	VICEC_RF_CHANNEL_POWER_RECEIVER_WRONG_MEAS_TYPE One of the two measurement types must be specified for each channel: Either single channel or multi channel measurement.
13010	VICEC_RF_CHANNEL_POWER_RECEIVER_CHANNELS_MUST_NOT_OVERLAP The CW API does not support overlapped channels at the time of this writing. That means that for each channel i with frequency f_i and bandwidth w_i the following condition must be true: $f_i + .5 * w_i < f_{(i+1)} - .5 * w_{(i+1)}$
13014	VICEC_CW_DETECTOR Returned if the detector is set to an invalid value (forced through an invalid cast). Can also be returned if the detector has not been set, but is required to perform another action (for example, the SetFrequencies() call requires the detector to be specified upfront).

12 ViCom GPS

As mentioned in previous chapters, the TSMW has a built-in GPS module. This chapter describes how to use this GPS receiver with ViCom.

ViCom GPS implements functionality which:

- can retrieve data from the GPS device
- send commands to the GPS device

The messages from the GPS device can be interpreted to provide information such as:

- GPS time
- Longitude
- Latitude
- Altitude
- Satellite information

Examples of the GPS functionality implemented in the ViCom are provided in this chapter.

The TSMW should be connected and configured as described in the [Appendix](#) (see Appendix A – TSMW Configuration). No extra configuration for the GPS is required.

The GPS data is provided over the same IP connection as the scanner data.

12.1 Sample Application

Below is a figure showing the ViCom GPS sample application.

As with other sample applications, the layout consists of several parts. On the left side are the general user controls used to control the application. The panel at the top allows the user to send commands to the GPS module and the middle panel shows the results.

The following section describes how to make measurements using the sample application and explains which ViCom functions are being used.

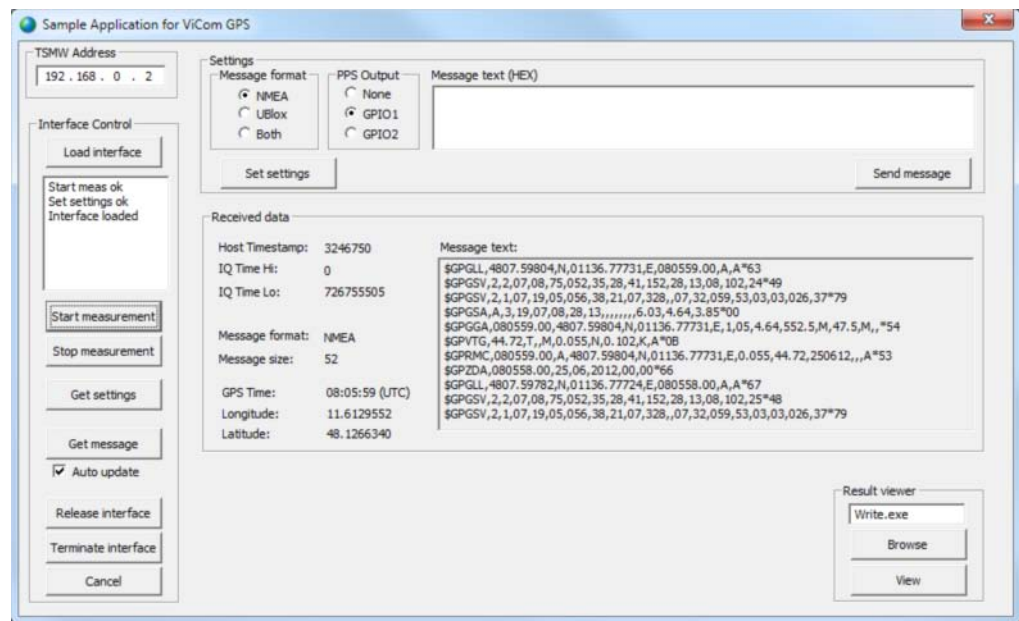


Figure 74 sample application for GPS

12.2 Measurements

Perform the following steps to make a GPS measurement.

12.2.1 TSMW Address

Enter the IP address of the TSMW in the given field (top left).

This generally has the form 192.168.0.x – where “x” is usually set to “2”.

After entering the address, press “Load Interface” to establish a connection with the TSMW.

12.2.2 Message format

Messages can have the NMEA format, Ublox format or both.

The format can be set either using the “Get format button” (which asks the GPS module what format it is currently configured for) or by selecting the radio button for the desired format and pressing “Set format”.

The dialog box next to the radio buttons allows you to send command messages to the GPS module. Obviously, messages should be sent in the given format.

Messages resulting from commands or from measurements, will appear in the message window. The messages begin with the command and are followed by the response – see [Commands](#) for examples.

12.2.3 Measurement

Pressing “Start measurement” starts the measurement process but does not generate any results.

Pressing “Get message” tells the application to take the next message in the buffer from the GPS module. The result is displayed in the “Message text” window in the “Received data” area.

Pressing “Stop measurement” ends the measurement process but does not release the interface.



Generating continuous measurement results

Selecting the “Auto update” check-box will generate continuous results after “Get message” is clicked.

To use this feature, the box must be checked before starting the measurement.

12.2.4 Commands

Commands can be entered as ASCII characters or hex codes depending on the message format. The interface does not check the validity of the commands you enter. Error messages resulting from incorrect or unsupported commands will be displayed in the message window.

In [Figure 74](#), the message window contains several commands and their results. Many such commands are available and some examples are explained in the following tables.

\$GPZDA	Date & Time	
	Format	\$GPZDA,hhmmss.ss,dd,mm,yyyy,xx,yy*CC \$GPZDA,201530.00,04,07,2002,00,00*60
	hhmmss	HrMinSec(UTC)
	dd,mm,yyy	Day,Month,Year
	xx	local zone hours -13..13
	yy	local zone minutes 0..59
	*CC	the checksum data, always begins with *

\$GPGSV	Satellites in View	
	Format	\$GPGSV,2,1,08,01,40,083,46,02,17,308,41,12,07,344,39,14,2 2,228,45*7
	2	Number of sentences for full data
	1	sentence 1 of 2
	08	Number of satellites in view
	01	Satellite PRN number

\$GPGSV	Satellites in View	
	40	Elevation, degrees
	083	Azimuth, degrees
	46	SNR - higher is better
	As above	For upto 4 satellites per sequence
	*75	the checksum data, always begins with *

\$GPGLL	Geographic position, Latitude and Longitude	
	Format	\$GPGLL,4916.45,N,12311.12,W,225444,A,*1D
	4916.46,N	Latitude 49 deg. 16.45 min. North
	12311.12,W	Longitude 123 deg. 11.12 min. West
	225444	Fix taken at 22:54:44 UTC
	A	Data Active or V (void)
	*iD	the checksum data, always begins with *

A complete list can be obtained from many sources on the internet.

12.2.5 Ending the session

After making your measurements you can unload the GPS receiver using the “Release interface” button.

If you encounter a problem during the release procedure, try using the “Terminate interface” button. The ViCom API then forces a deletion of the `ViComGPSInterface` object.

12.2.6 Viewing the results

To view the results, use the controls in the lower right corner.

Clicking “Browse” lets you find the result file you want to open, and “View” opens the file in the viewer.

13 RS232 Tunneling

The TSMx series has different connection ports at the back: Two firewire connections, several antenna connectors, a power connector and one RS232 interface. The latter can be used to connect an additional device to the TSMx, making it controllable by the PC that the TSMx is connected to.

RS232 communication supports two-way communication, as long as it is not done parallel. For example, some older mobiles can be connected by a serial connector to the pc. In general, any device that supports a serial communication interface (RS232) can be connected and controlled using the ViCom API.

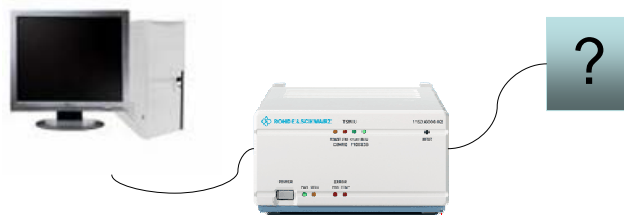


Figure 75 RS232 system configuration

In this chapter, the API of the ViCom package is described that covers the area of communicating with the connected device. Since there are no special measurement modes in general that can be described, this chapter only contains two sections:

The next section contains a brief walk-through example of the sample application which is part of the delivery. This is followed by a short demonstration how to use the API in the code.



Please make sure that the firmware version 12.xx or higher is installed on the TSMx device. You can do this in the sample application after the device has been loaded. Check the Firmware Version property shown in the list below the “Load Device” button.

Firmware version less than 12.xx don't support the RS232 API. In the sample application, you encounter a “Selected TSMx does not support the desired measurement” message when clicking on the “Start Measurement” button. Refer to the 13.2D for a description how to update the TSMx firmware.

13.1 Sample Application

Below you can a screenshot of the RS232 sample application. Its layout structure is taken from the other sample applications. The left side contains the general control buttons that are used to set the TSMx in a desired state or to apply settings.

As in the other applications, the center of the dialog is consumed by the technology specific elements. In this case, this is only a text field in which data can be entered. Using the button “Send Data” below the text field, the data can be transmitted via the TSMx to the connected device.

The right side contains the configuration elements. In this case, two settings can be modified:

- The baud rate: This determines the through-put of the connection. Normally, one value of the following list fits for most devices: 4800, 9600, 14400, 19200, 38400, 57600 or 115200.
- Silent mode (marked in the screenshot with a green ellipsis): This specifies whether the silent mode is switched on (4) or off (2). A value of 0 means that the current configuration is not changed. Refer to the section below for more details.

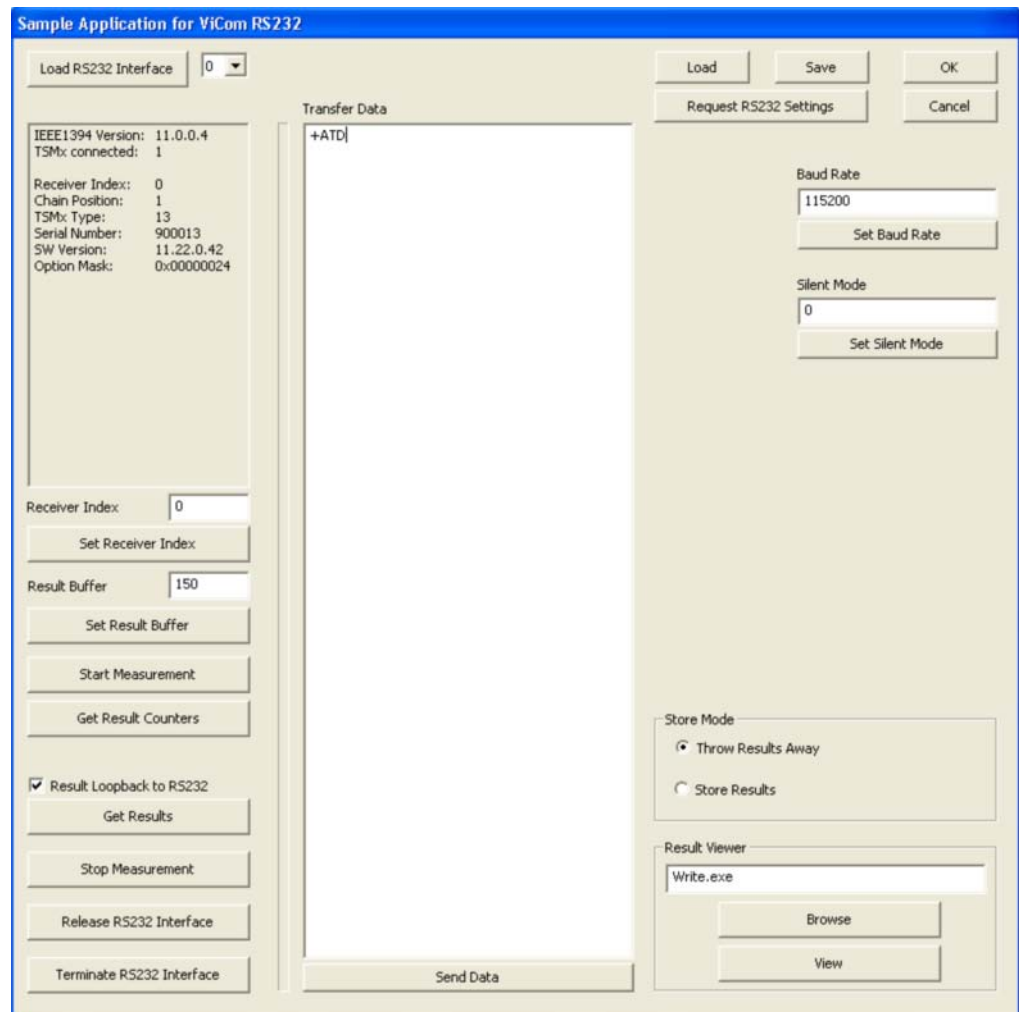


Figure 76 Sample Application for RS232 API

13.1.1 Walk-Through Example

The general procedure to set up a “measurement” with the RS232 API is as follows:

1. Press button “Load RS232 Interface”
2. Check if an appropriate firmware version is loaded (Firmware Version \geq 12.x)
3. Press “Set Receiver Index” button to specify which device shall be controlled in the firewire chain.
4. Set the result buffer size with the according button.
5. Set Baud Rate and Silent Mode if desired with the buttons on the right side of the dialog.
6. Press “Start Measurement” button
7. Perform you work (see sections below for details).
8. Press “Stop Measurement” button. You can now change some settings and restart the measurement (see step 6).
9. Press “Release Measurement” button. You can now close the application or load another device (see step 1).

13.1.1.1 Sending Data

Now the communication with the serial port can be started. Data can be sent by entering text into the central text field and send it to the connected device by pressing “Send Data”.

13.1.1.2 Getting Data

Data can be fetched by clicking on “Get Results”. The sample reads data until no further data can be found in the result buffer.

To fill the result buffer, the ViCom implementation if the RS232 tunnelling API starts a thread to repeatedly watch the data sent on the firewire connection. If it finds data from the serial port of a TSMx device, it stores the data in the result buffer. If there is no such data found in one second, an empty result is stored.



Therefore, the result counters constantly increases, even if there is no communication at all. Be aware when using the API that after a call to `GetResult()` succeeded, the returned result might not necessarily contain valid information.

If there is data in the result buffer, its content is written to the output file “ViComMeasurements.txt” when the “Get Result” button is pressed and the “Store Results” mode is set. Additionally, the raw data that is received is written into another text file called “ViComMeasurements_RS232Received.txt”.

13.1.2 Result Loopback to RS232

The sample application supports a special test mode, in which incoming data is immediately sent back to the transmitter. For example, this is useful when a terminal is connected to the COM port of the TSMx (like the HyperTerminal application shipped with Microsoft Windows). Data sent by the terminal is then bounced back to have some kind of debug facility.

To enable that special mode, just check the “Result Loopback to RS232” button in the application, which is marked in the screenshot shown above with a red ellipsis.



This only works in the measurement mode. So be sure to check that box before you click on the GetResults() button. During the receiving of the results, this will force them back.

13.1.3 Silent Mode

Normally, the TSMx uses the integrated COM-Port to show trace messages that can be used to check about the current state of the device. In the RS232 measurement mode, this is automatically switched off. If you also want to turn that messages of when no RS232 API is active, set this value in the TSMx persistently (enter mode, press “Set Silent Mode” button and “Start Measurement”; only if the measurement has been started once after the mode was changed, the change will take effect).



The silent mode will be stored as setting in the TSMx device itself. Even after releasing the ViCom interface or after switching the device off and on, this setting is restored in the next session. Therefore the value of 0 can be specified as well, indicating that the current setting shall not be changed.

When the silent mode is activated, the TSMx indicates that mode by letting the Process/State LED blink constantly (when the interface is loaded and no measurement is active).

13.2 Sample Code

As for all other chapters, this one also ends with a code sample. The structure is similar to the ones listed in the other chapters. The main goal is the same: Show the usage of the API in a simple environment to focus on the usage of the API.

The purpose of the sample is to constantly read data from the result buffer and print it to the console if data is available. To send data, the user can start typing and finish his input by pressing RETURN. When the user wants to exit the application, pressing ESC shall do so.

```

#include "stdafx.h"

#include <windows.h>
#include <iostream>
5 #include <conio.h>

#include "ViComBasicInterfaceData.h"
#include "ViComError.h"
#include "ViComLoader.h"
10 #include "ViComRs232Errors.h"
#include "ViComRs232Interface.h"
#include "ViComRs232InterfaceData.h"

using namespace std;

15
/*****

#define EXEC_VICOM( msg, call ) \
20   if ( strlen(msg) ) { cout << msg << "..."; } \
   call; \
   if ( cError.GetErrorCode() > 0 ) { \
   char* szErr = new char[strlen(cError.GetErrorString()) + 1]; \
   strcpy(szErr,cError.GetErrorString()); \
25   throw CViComError(cError.GetErrorCode(), szErr); } \
   if ( strlen(msg) ) { cout << "ok" << endl; }

/*****

void RunRs232Sample()
30 {
   CViComError cError;
   CViComLoader< CViComRS232Interface > cRs232Loader;

   EXEC_VICOM( "Loading RS232 Tunnelling API",
35     cRs232Loader.LoadTsmx( cError ) );

   EXEC_VICOM( "Retrieving special interface",
     CViComRS232Interface* pcRs232 = cRs232Loader.GetpInterface( cError ) );

40   EXEC_VICOM( "Start measurement",
     pcRs232->GetBasicInterface().StartMeasurement( cError ) );

   cout << "Press ESC to quit application..." << endl;

45   while ( true )
   {
     // User interface is as follows: Listen to the "virtual COM"
     // device and print incoming data until a key is pressed. If the
     // key is the ESC character, the application is stopped. Otherwise
50     // the input string is used to send a command to the connected
     // COM device. If the input string is empty, the application is
     // stopped as well.
     if ( !_kbhit() )
55     {
       CViComRS232InterfaceData::SMeasResult* pcMeasResult;

       pcMeasResult = pcRs232->GetResult( cError, 100 );
       if ( ! pcMeasResult && cError.GetErrorCode() != 119 )
60     {
         char* szErr = new char[strlen(cError.GetErrorString()) + 1];
         strcpy(szErr,cError.GetErrorString());
         throw CViComError(cError.GetErrorCode(), szErr);
       }

65     // Continue if no result is available or no actual result
     // is in the measurement result. This can be the case since
     // the internal watchdog thread listening on the firewire
     // connections for results of the COM interface generates
     // "empty results" in each turn (one every second).
70     if ( ! pcMeasResult || ! pcMeasResult->pReceiveData )
     {

```

```

        continue;
    }

75     //
        // Print the result data, character by character to include
        // control symbols and NULL characters that belong to the
        // result as well.
        //
80     cout << "Data received: ";
        for ( int i=0; i<pcMeasResult->pReceiveData->dwCharCount; i++ )
        {
            cout << pcMeasResult->pReceiveData->pcCharacters[i];
        }
85     cout << endl;

        continue;
    }

90     // This is the command center: Exit the while loop if the user
        // pressed ESC or the input string read is of zero length.
        char szCommand[ 1024 ];
        szCommand[0] = getch();
95     if ( szCommand[0] == 27 || szCommand[0] == 13 )
        {
            cout << "Pressed ESC or entered empty string => quit demo" << endl;
            break;
        }

100    cout << szCommand[0];
        _cscanf( "%s", &szCommand[1] );

        // consume final return key
105    getch();

        CViComRS232InterfaceData::SCharacterBuffer sCharacterBuffer;
        sCharacterBuffer.dwCharCount = strlen( szCommand );
        sCharacterBuffer.pcCharacters = szCommand;
110    EXEC_VICOM( "Sending data",
                pcRs232->SendData( cError, sCharacterBuffer ) );
    }

115    EXEC_VICOM( "Stop measurement",
                pcRs232->GetBasicInterface().StopMeasurement( cError ) );

        EXEC_VICOM( "Waiting for stop signal",
                pcRs232->GetBasicInterface().HasMeasurementStopped( cError, 1000 ) );
120    EXEC_VICOM( "Unloading...",
                cRs232Loader.ReleaseTsmx( cError, false ) );
    }

125    /*****

int _tmain(int argc, _TCHAR* argv[])
{
130    try
    {
        cout << "ViCom RS232 Scanner Demo" << endl << endl << endl;

        RunRs232Sample();
    }
135    catch ( CViComError& cError )
    {
        cout << cError.GetErrorString() << endl;
        delete [] cError.GetErrorString();
        return cError.GetErrorCode();
140    }

    return 0;

```

```
}
```

The setup procedure is the same as in every other ViCom sample application. Even the configuration of the measurement is kept minimal in this sample, no special settings are defined. Instead, default values are used (115200 for the Baud Rate) and the application is started in line 41.

The most interesting stuff happens inside the while loop. The loop is an endless loop that can only be exited in line 98, which is executed if the user presses ESC or enters an empty command string.

In lines 58-86, the results are read from the device and then printed to console, if information is available. One important aspect shown in the example is that the error returned from the `GetResult()` method is not directly interpreted by the `EXEC_VICOM()` macro, but it is checked manually. This is done since the RS232 API does deliver results only every second. If no result can be found in the given timeout-window, such a timeout is returned as error. To handle that case, the error code is checked.

Printing the result is a simple task and is done in line 80-85.

The program control logic is performed in lines 93 to 112. The first character is read as first part of the command string, if it is not an exit key (lines 93-99). Then the rest of the command is read using the `scanf()` function. The command is terminated by a RETURN character, which is consumed separately since the `scanf()` function does not consume that character (lines 101-106).

Once the command string is found, it is send using the ViCom interface.

Appendix

A TSMW Configuration

The TSMW does not come with a desktop utility to install software options on the device or to perform firmware updates. It offers a configuration web page similar to the configuration tools of DSL routers, for example. That page can be found when entering the IP address of the device into the address bar of a web browser (192.168.0.2 is the default address).

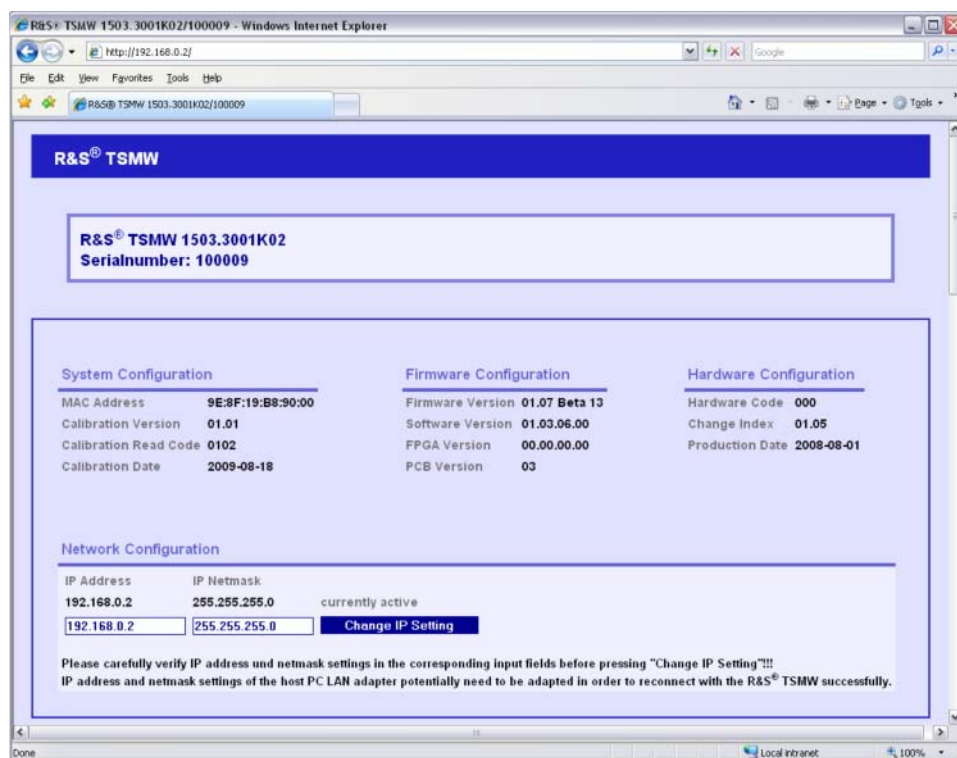
In the following sections, a short overview of the functionality and information offered within that web page is described. This must be seen as a quick reference on that topics. For a more decent description of these issues, the TSMW User Manual must be consulted (see [3]).

A.1 System Information

If the web page can be opened, it should look similar to the one depicted below. There are several basic information shown on the first page, like the software version numbers and the current IP configuration.



Within that page, the IP address can be changed. Please be aware that changing the IP address might require additional changes on the host PC resp. might also disable the possibility to connect to the TSMW at all (if the IP address is always in use, for example).



A.2 Option Handling

The options currently installed on the TSMW device are listed in the “Active Optionkeys” section of the page. An additional section lists the options that were once active but have been disabled in the meantime.

Option Index	Option Type	Material Number	Optionkey	Format	Privilege	Timestamp	License Count	Activation Type	Valid From	Valid To	Time To Expiration
1	TSMW- K1	1503.3960.02	136969178412034997631573236926	0	Customer Order	2008-07-03 06:30	1	Permanent	-	-	-
21	TSMW- K21	1503.4514.02	356866547527413676630206567647	0	Customer Order	2009-07-29 06:18	1	Permanent	-	-	-
22	TSMW- K22	1503.4520.02	205947346541342637510377498559	0	Customer Order	2009-07-29 06:18	1	Permanent	-	-	-
26	TSMW- K26	1510.8792.02	016053306928060559851401349648	0	Customer Order	2009-07-29 06:18	1	Permanent	-	-	-
27	TSMW- K27	1503.4537.02	226826789119888685561107045374	0	Customer Order	2009-10-20 09:26	1	Permanent	-	-	-
28	TSMW- K28	1503.4543.02	258331664939731883060795245924	0	Customer Order	2008-07-17 07:52	1	Permanent	-	-	-
29	TSMW- K29	1503.4550.02	015430682406919203092337151146	0	Customer Order	2009-03-18 06:42	1	Permanent	-	-	-
121	TSMW- K121	1503.4589.02	311757375413324347150953894928	0	Customer Order	2009-07-29 06:18	1	Permanent	-	-	-
122	TSMW- K122	1503.4595.02	245543933616086293802786216943	0	Customer Order	2009-07-29 06:18	1	Permanent	-	-	-
126	TSMW- K126	1503.4789.02	159963900903968861720640471947	0	Customer Order	2009-07-29 06:18	1	Permanent	-	-	-
127	TSMW- K127	1503.4614.02	2541815906395339205728223689360	0	Customer Order	2009-10-20 09:26	1	Permanent	-	-	-
128	TSMW- K128	1503.4566.02	295036083233863879230626211017	0	Customer Order	2008-07-17 07:52	1	Permanent	-	-	-
129	TSMW- K129	1503.4572.02	397169887811946762342241135434	0	Customer Order	2009-03-18 06:42	1	Permanent	-	-	-
221	TSMW- K221	1503.4795.02	084823431638520694222366864086	0	Customer Order	2009-07-29 06:18	1	Permanent	-	-	-
222	TSMW- K222	1503.4808.02	053019402304397846903845663196	0	Customer Order	2009-07-29 06:18	1	Permanent	-	-	-

Option Index	Option Type	Material Number	Optionkey	Format	Privilege	Timestamp	License Count	Activation Type	Valid From	Valid To	Inactivity Reason
0	TSMW- K0	1514.4327.02	380469971912113853701473899089	0	Demo	2010-02-24 14:12	1	Temporary Date	2010-02-24	2011-02-24	not yet in activation phase

New options can be installed when either an option file or an option key is available. This can be done in the “Install Option Key” section shown below.

xml-file Browse... Install

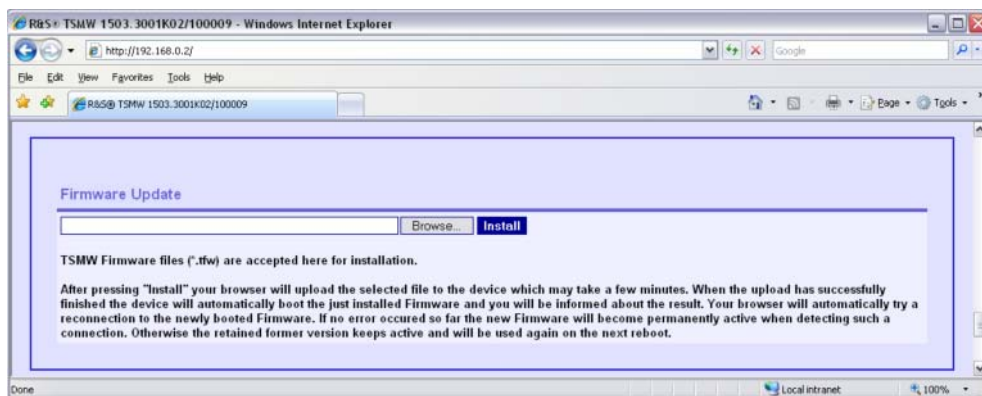
enter manually Install

R&S® TSMW option order(s) are shipped as *.xml files on a CD-ROM. For option installation, insert the CD-ROM into the computer drive, press the "Browse" button, select the corresponding option file on the CD-ROM and press "Install". Alternatively to xml-file input, the option key could be entered manually.

After finishing installation this page is being updated automatically and the added option should be listed in the instrument options section.

A.3 Firmware Update

The TSMW firmware can be updated using the same HTML Page where the options can be queried. In the section “Firmware Update” the “Browse...” button can be used to search for the firmware file that contains the new firmware code. Once the path to the file is visible in the text field, the “Install” button can be used to start the update of the TSMW firmware.



B TSMx Option Handling

With the TsmxOptionKeyInstaller utility, it is possible to check the IEEE1394 connection and check the instrument data set. Additionally, the program displays all the enabled options for a certain instrument and can be used to install new options. The instrument specific Device Identification file can be read out.

This tool can be started from the Windows start menu:

“Rohde & Schwarz -> ViCom -> Tools -> TsmxOptionKeyInstaller”

If the log messages of the OptionKeyInstaller show that a connected R&S TSMx device has been found, this means that the IEEE1394 connection is working, and that the ROMES demo or the test application can be run. The OptionKeyInstaller utility can also be used to recall instrument data, to verify the installed options and to install new options if required.

When you are satisfied that the IEEE1394 connection is working, disconnect the OptionKeyInstaller from the R&S TSMx and close the utility.

B.1 Prerequisites

- PC and R&S® TSMx connected via IEEE1394
- PC and R&S® TSMx connected via a serial null-modem cable
- R&S proprietary IEEE1394 driver installed on the PC
- Terminal program started
- No other instrument interfacing the R&S TSMx

B.2 Program Start

To use this program, reset the R&S® TSMx and wait for the terminal output “**Waiting for elf file from IEEE1394.**”

Start the TsmxOptionKeyInstaller utility by executing the TsmxOptionKeyInstaller.exe in the program directory. After establishing the connection, a dialog similar to the one shown below comes up.

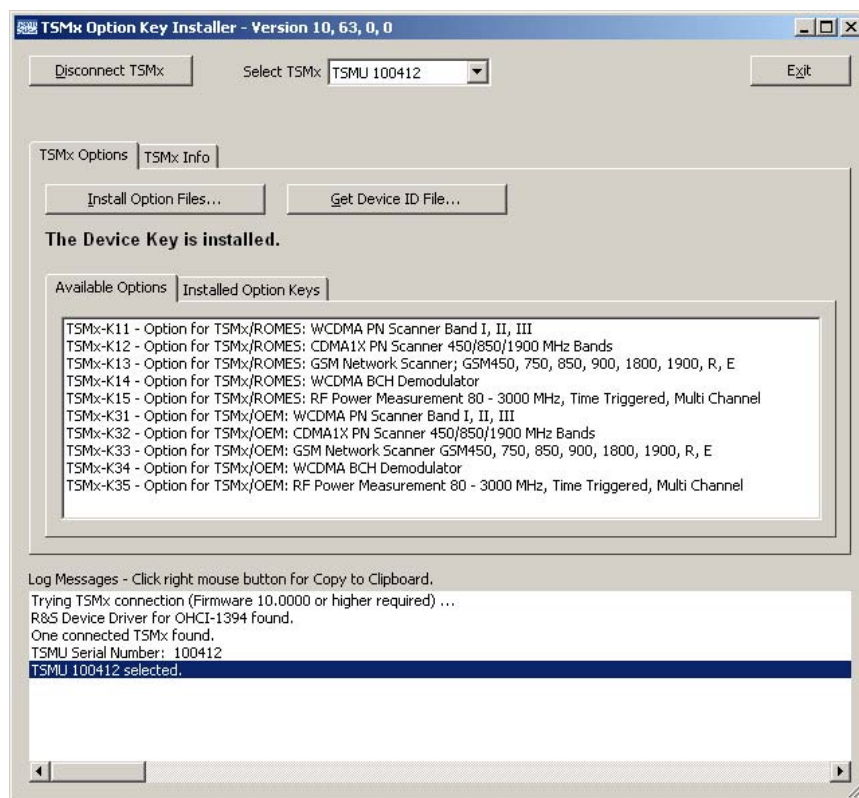


Figure 45 TsmxOptionKeyInstaller Info Tab

B.3 Display Contents of the R&S TSMx Info Tab

The following information can be read out from the R&S TSMx Info tab:

Hardware

- Type instrument type
 R&S TSMU VAR 02
 R&S TSMU-H VAR 03
 R&S TSML-x x has to be identical with the label at the rear panel!
- Input Power type of input stage Normal: standard input power (R&S TSMU, R&S TSML-x)
 High: extended input power (R&S TSMU-H)
- JTAG Devices ACE V2Pro: 1st Controller Board version
 ACE V2Pro Counter: 2nd Controller Board version (with CPLD)

- IEEE1394 Chip Rev.: revision of the OHCI1394 chip version has to be $> = 0x61$

Serial Numbers

Programmed serial numbers in the EEPROM and in the flash card.

The following two serial numbers should be identical:

- TSMx(Flash): R&S TSMU serial number that is stored on the flash card
- TSMx(EEPROM): R&S TSMU serial number that is programmed in the EEPROM

The following two serial numbers should be identical (but not necessarily the same as the R&S TSMU serial number above!):

- Receiver(Flash): Front-End serial number that is stored on the flash card
- Receiver(EEPROM): Front-End serial number programmed in the Front-End EEPROM

Firmware

- Virtex2Pro version of the FPGA configuration data and boot code e.g. H01**V10B0**
H01: 2nd Hardware version Controller Board
(H00: 1st Hardware Version Controller Board
V10: Firmware version 10
B0: Beta state 0
- Application version and file data of the TxmsOptionKeyInstaller utility application file *.elf
- Firewire Loader version and file length of the Firewire Loader program
- Flash Loader version and file length of the Flash Loader program file

Calibration

- Level Level calibration file over the full input frequency range of the R&S TSMx
- IF-Filter IF-Filter calibration file over the full input frequency range of the R&S TSMx
- WCDMA 3GPP Derived calibration data set of the IF-Filter (only for WCDMA 3GPP measurements)

Level- and IF-Filter files are created during calibration process. The WCDMA 3GPP IF-Filter correction file is stored when the UMTS PN Scanner Application is started for the first time after calibration.

B.4 Display Contents of the R&S TSMx Options Tab

The “**Available Options**” list box in the “**R&S TSMx Options**” tab gives an overview of all the enabled options for a particular instrument.

The “**Installed Option Keys**” list box displays all the installed OptionKeys and the expiry date of time- limited keys.

If that list is empty and the program indicates “**No device key has been found**” (circled in red in Figure 77 below) all the available options are enabled with application file coding.



Instruments without installed device key require the installation of option keys!

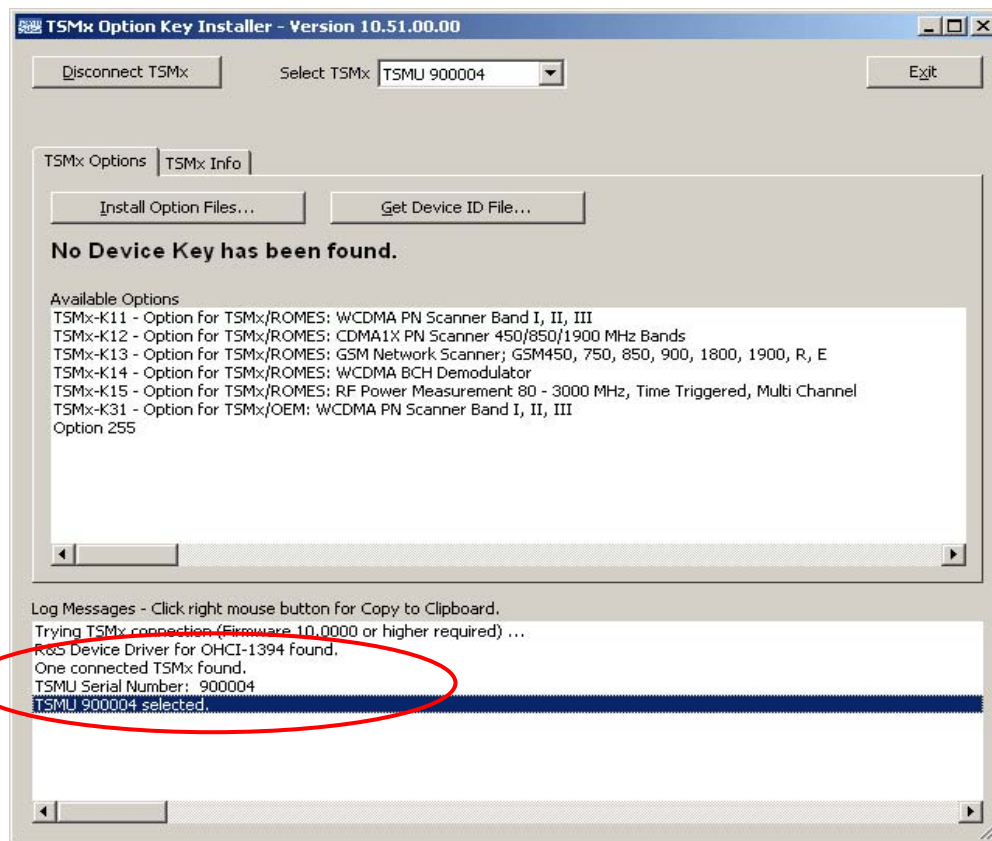


Figure 77 TsmxOptionKeyInstaller Options Tab

Installed Options for measuring in different technologies, using different R&S TSMx devices should be as follows:

Phylis modules	R&S TSML-W	R&S TSMU
W-CDMA 3GPP measuring without BCH Demodulator:		
ViComWCDMA3GppPns.dll	11	31 (locked)
W-CDMA 3GPP measuring with BCH Demodulator:		
ViComWCDMA3GppPns.dll UmtsDemodulator.dll	11, 14	31, 34 (locked)
CDMA2000 measuring:		
ViComCdma2000Pns.dll	12	32 (locked)
GSM Network Scanner measuring:		
ViComGsmNws.dll	13	33 (locked)

Program Exit

The “**Disconnect TSMx**” button of the TsmxOptionKeyInstaller utility can be used to stop the connection with the connected instrument(s). This will initialize every connected R&S® TSMx to reboot.

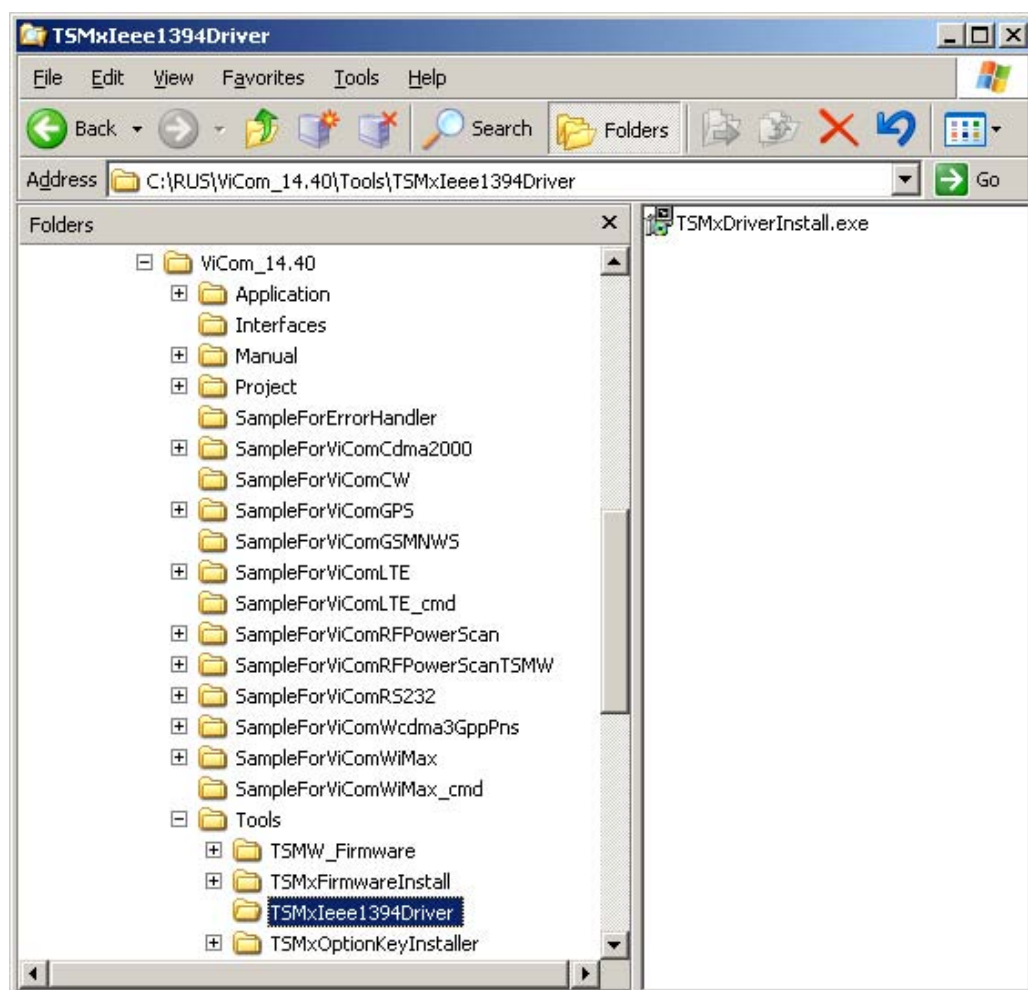
Afterwards, press “**Exit**” to quit the program or “**Connect TSMx**” to connect the instruments again.

C Installing the TSMx Windows device driver manually

When the ROMES demo application supplied on the CD-ROM is installed, the TSMx Windows device driver is automatically installed. Therefore, if ROMES demo has been installed on your computer, there is no need to follow the instructions in this section.

If ROMES demo has not been installed on your computer, then you will need to install the Windows device driver manually, before a connection to a TSMx can be made.

If the ViCom.zip file was unpacked onto your C:\ drive, the executable TSMXDriverInstall.exe will be found in the following location:



Double-click on the executable “TSMxDriverInstall.exe.”

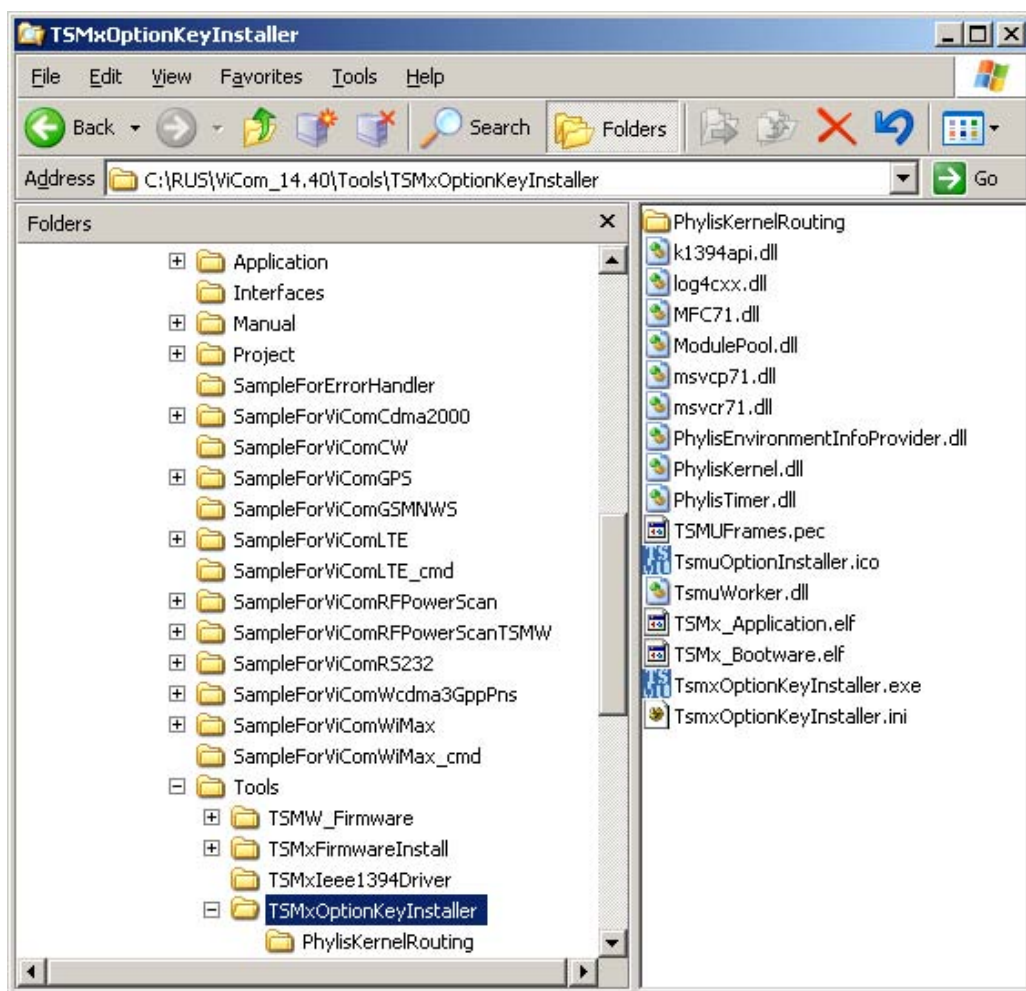
After a short pause, the following splash screen appear.



Follow the instructions in the dialogue box, and make sure that no application is using a TSMx, and that the Device Manager is not displaying TSMx Property pages.

Click “Next” to start installing the device driver, and follow the instructions in the install wizard.

When the device driver is installed, the Firewire connection to the TSMx may be checked with the OptionKeyInstaller utility supplied in the ViCom zip file on the CD-ROM. If the zip file was unpacked as described earlier in this document, the OptionKeyInstaller may be found in the following location:



Instructions for using TsmxOptionKeyInstaller are given in Appendix B TSMx Option Handling on page 232 of this document. If the log messages of the OptionKeyInstaller show that a connected R&S TSMx device has been found, this means that the IEEE1394 connection is working, and that the ROMES demo or the test application can be run. The OptionKeyInstaller utility can also be used to recall instrument data, to verify the installed options and to install new options if required.

When you are satisfied that the IEEE1394 connection is working, disconnect the OptionKeyInstaller from the TSMx and close the utility.

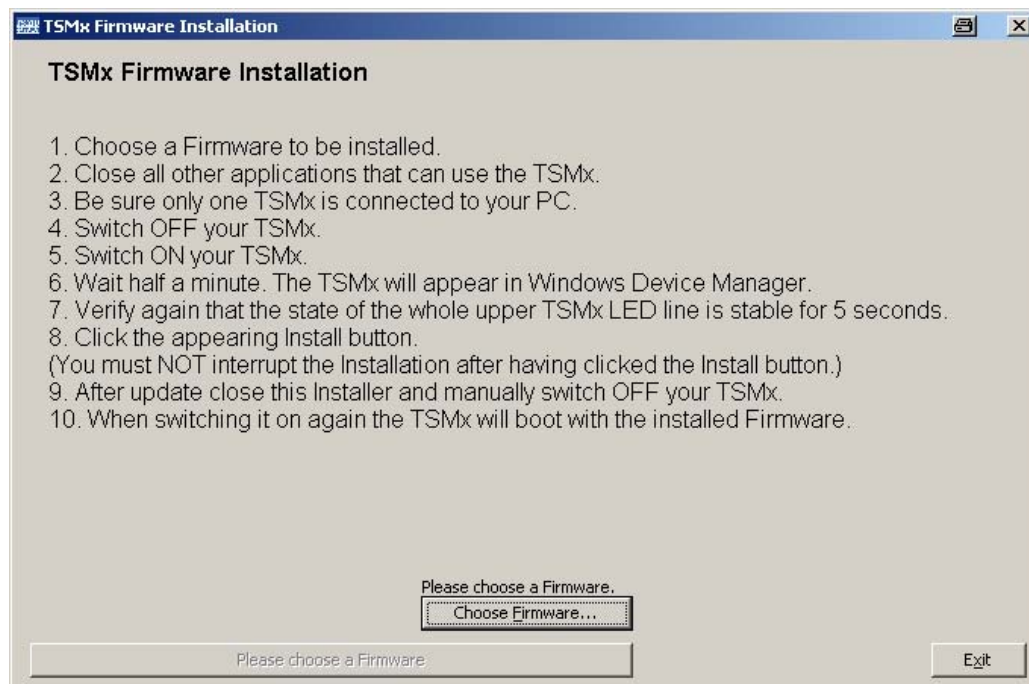
D TSMx Firmware Upgrade

For some special functions, a new firmware must be installed on the TSMx device. Therefore, a special tool called "TsmxFirmwareInstall.exe" can be used.



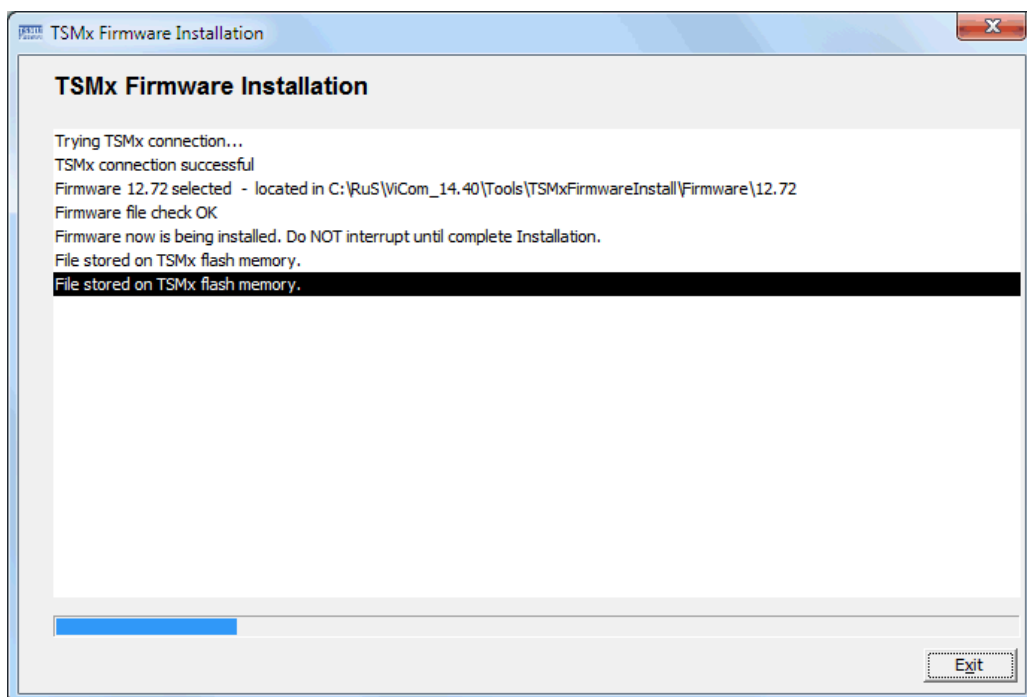
If you do a firmware installation, make sure to follow ALL listed steps in this manual CAREFULLY. If you miss any single detail, this might cause the firmware installation to fail and leave an unusable device up to you.

After you started the tool, you should see a window similar to the one shown below:



The installation is then done as follows:

1. Choose the firmware that shall be installed. Click on the button marked with a blue ellipsis. A drop-down menu opens and lists all the available firmware versions. These are extracted from the Firmware subfolder that can be found in the application path. Selected the appropriate version.
2. Close all other applications that might use TSMx. These are the sample applications, the option installer tool and your own TSMx-related applications.
3. Check if only one TSMx is connected to the PC. The tool does not check how many instances of TSMx device are connected with each other and the PC. Since the data is transferred on the firewire connection without a specific destination, behavior is undefined if multiple devices are connected to the PC during installation.
4. Switch TSMx off and on.
5. Wait about 30 seconds until the TSMx LEDs stopped blinking. After that time, the LEDs should not blink at all or show a constant behavior for at least 5 seconds. If the Process/State LED keeps blinking, the silent mode of the RS232 API might still be turned on. In that case, a firmware update can be started.
6. The installation button in the screenshot above can now be pressed. A progress window is shown now.



7. After the installation finished, reboot your TSMx device manually to really apply the update to all parts of the software.



After the update has finished, it is essential to reboot the TSMx. Otherwise some new functionality might not work correctly.

E Frequently Asked Questions

- ? *Does the ViCom system take advantage of multi-core PC architecture?*
- ! Yes, ViCom is designed internally to use multiple threads for different measurement tasks, so more CPUs result in a “more parallel” execution of the measurement processing algorithms.
- ? *When I run the sample application, it asks for dlls that are not listed in Section 2 of this document, and are not present on the CD-ROM*
- ! Check that all dlls in the Application sub-directory are writeable as well as readable (Read Only box of the Properties is not checked).
- ? *When I click “Load PNS Scanner” in the sample program, I get an error message “The interface version of the ViCom library and the interface header file does not match.”*
- ! The interface version corresponds to the last part of the file version string of the ViCom interface dll, named ViComXXXX.dll, where XXXX specifies the interface type. For example, if the file version of the interface dll, ViComWCDMA3GppPns.dll is 10.70.0.18 then the interface version of the DLL is 18.

The interface version of the DLL has to match the interface version that is defined in the corresponding ViCom header files. If the version does not match then the interface loader will fail and will produce a ViCom Error. In this case, please contact Rohde & Schwarz Technical Support.

You can view the version of the interface dll by right clicking on the dll in Windows Explorer, choosing Properties, and the Version tab.

- ? *When I click "Start Measurement" in the RS232 application, I get the message "Hardware: Selected TSMx does not support desired measurement." What is wrong here?*
- ! The firmware version does not support the RS232 application. Please make sure to have a firmware version of 12.x.x.x or higher installed on the TSMx device. Refer to the firmware installation section to update your device.
- ? *Why do I always get a message "Connection lost" when I try to debug the ViCom sample application or my own ViCom using programs?*
- ! The communication between the device and the internal processing modules assumes to have very little timeouts. When the processing of an application is paused (for example, because a breakpoint is hit in the debugger), chances are high that the timeout is reached before execution is continued. Use the "TsmuWorkerDebugSettings.reg" file in the LogFiles to increase the timeout and leverage that effect in your debug environment. Please be aware that this also changes the behavior in released versions, so be sure to disable that setting again when doing tests in the release version. Refer to chapter Debugging Techniques.

F UMTS Technical Notes

F.1 UMTS Channels

The relationship between UMTS UARFCNs (UTRA Absolute Radio Frequency Channel Numbers) N and carrier frequencies F is defined in Reference 6, Section 5.

From Reference 6, UARFCN is calculated according to the following formula:

$$N = 5 * (F - F_{\text{Offset}}),$$

The frequency offset F_{Offset} is defined in Reference 6 for each UMTS band, as well as for the extra frequencies assigned in Band II.

The downlink and uplink channels assigned in the operating bands I, II and III are listed in the tables below. Note in operating band II, the 12 additional centre frequencies that are specified, shifted by 100 kHz relative to the normal raster .

The downlink and uplink channels assigned in the operating bands I, II and III are listed in the tables below. Note that in operating band II, 12 additional center frequencies are specified. These additional channels are shifted by 100 kHz relative to the normal raster and not calculated according to the formula above.

F.2 UTRA operating bands and channel numbers: Downlink

Operating Band	DL Frequency Band	Frequency offset F_{Offset}	Assigned Channels	Assigned Center Frequencies
I	2110 MHz to 2170 MHz	0	10562 to 10838	2112.4 MHz to 2167.6 MHz in steps of 0.2 MHz
II	1930 MHz to 1990 MHz	0	9662 to 9938,	1932.4 MHz to 1987.6 MHz, in steps of 0.2 MHz
Extra downlink channels of Band II	as for Band II	1850.1	412, 437, 462, 487, 512, 537, 562, 587, 612, 637, 662, 687	1932.5, 1937.5, 1942.5, 1947.5, 1952.5, 1957.5, 1962.5, 1967.5, 1972.5, 1977.5, 1982.5, 1987.5
III	1805 MHz to 1880 MHz	1575	1162 to 1513	1807.4 MHz to 1877.6 MHz in steps of 0.2 MHz

F.3 UTRA operating bands and channel numbers: Uplink

Operating Band	UL Frequency Band	Frequency offset F_{Offset}	Assigned Channels	Assigned Center Frequencies
I	1920 MHz to 1980 MHz	0	9612 to 9888	1922.4 MHz to 1977.6 MHz in steps of 0.2 MHz
II	1850 MHz to 1910 MHz	0	9262 to 9538,	1852.4 MHz to 1907.6 MHz in steps of 0.2 MHz
Extra uplink channels of Band II	as for Band II.	1850.1	12, 37, 62, 87, 112, 137, 162, 187, 212, 237, 262, 287	1852.5, 1857.5, 1862.5, 1867.5, 1872.5, 1877.5, 1882.5, 1887.5, 1892.5, 1897.5, 1902.5, 1907.5
III	1710 MHz to 1785 MHz	1525	937 to 1288	1712.4 MHz to 1782.6 MHz, in steps of 0.2 MHz.

F.4 System Information Blocks

UMTS System Information Blocks (SIBs) are emitted on the Broadcast Channel (BCH). They contain information that is likely to be relevant to any phone in the cell. SIBs and their contents are defined in Reference 6, the 3GPP RRC protocol specification. The table below shows a summary.

The Corresponding PDU column refers to the ViCom interface parameter used to specify which SIB(s) are to be demodulated by the BCH demodulator of the R&S TSML-W. See also Section 6 “SChannelPDUPair” above.

SIB Number	Contents	Reference in 3GPP TS 25.133v5.x.x	Corresponding PDU
3	NAS system information. UE timers and counters.	10.2.48.8.4	15
3	URA Identity	10.2.48.8.5	16
3	Parameters needed for cell selection and reselection, including the identity of the current cell.	10.2.48.8.6	17
4	Parameters needed for cell selection and reselection when in connected mode.	10.2.48.8.7	18
5	Parameters for the configuration of the common physical channels in the cell.	10.2.48.8.8	19
6	Parameters for the configuration of the common and shared physical channels to be used in connected mode.	10.2.48.8.9	20
7	Two fast changing parameters; UL interference and Dynamic persistence level.	10.2.48.8.10	21
8	not defined		22
9	not defined		23

SIB Number	Contents	Reference in 3GPP TS 25.133v5.x.x	Corresponding PDU
10	not defined		24
11	Measurement control information for the cell that is broadcasting the SIB.	10.2.48.8.14	25
12	Measurement control information to be used in connected mode.	10.2.48.8.15	26
13	ANSI-41 system information	10.2.48.8.16	27
13.1	ANSI-41 RAND information.	10.2.48.8.16.1	28
13.2	ANSI-41 User Zone Identification information.	10.2.48.8.16.2	29
13.3	ANSI-41 Private Neighbour List information.	10.2.48.8.16.3	30
13.4	ANSI-41 Global Service Redirection information.	10.2.48.8.16.4	31
14	Parameters for common and dedicated physical channel uplink outer loop power control information to be used in both idle and connected mode.	10.2.48.8.17	32
15	Assistance information for UE-based or UE-assisted positioning methods	10.2.48.8.18	33
15.1	Assistance data for Differential GPS measurements.	10.2.48.8.18.1	34
15.2	Assistance data for A-GPS measurements	10.2.48.8.18.2	35
15.3	Assistance data for A-GPS measurements.	10.2.48.8.18.3	36
15.4	Ciphering information and assistance data for OTDOA positioning.	10.2.48.8.18.4	37
15.5	Assistance data for OTDOA positioning.	10.2.48.8.18.5	38
16	Radio bearer, transport channel and physical channel parameters to be stored by UE in idle and connected mode for use during inter-system handovers.	10.2.48.8.19	39
17	fast changing parameters for the configuration of the shared physical channels to be used in connected mode (UMTS TDD systems only).	10.2.48.8.20	40
18	PLMN identities of neighbouring cells, used in idle mode as well as in connected mode.	10.2.48.8.21	41

Index

C

Conventions	7
CPICH Channel Impulse Response (CIR) Measurements.....	44

D

Diagnostics Information	22
Display Contents of the R&S TSMx Options ..	234

G

General Description	10
Get the Firewire connection working	20

H

Hardware.....	14
---------------	----

I

index entry	
subentry	10
Install the ViCom interface and test application, including the OptionKeyInstaller utility.....	16
Installing ROMES demo and the Test Application	16

M

Managing more than one R&S TSMx scanner .	23
---	----

P

Prerequisites:.....	232
Program.....	236
Program Start.....	233

R

Releasing the Scanner.....	31
----------------------------	----

S

Software.....	15
Start Programming.....	27
Synchronising the R&S TSMx internal clock from an external source.....	15

T

Table 6 LTE Demodulation Modes	See
Troubleshooting Using the TsmxOptionKeyInstaller Utility.....	232

U

UMTS Channels.....	241
Using ROMES demo application	20

V

ViCom Sample Application.....	45
ViCom WCDMA Interface Files	17
ViComWCDMA3GppPns interface files	18