R&S®TSMW Interface & Programming Manual R&S®TSMW-K1

Software Manual

1503.3776.32 – 04

Test & Measurement

Software Manual

ROHDE&SCHWARZ

The Software Manual describes the following R&S®TSMW Interface options:

- R&S®TSMW-K1 – GigaBit Digital I/Q Interface (1503.3960.02)

The following abbreviations are used throughout this manual:
R&S® TSMW-K1 is abbreviated as R&S TSMW-K1.
Radio Network Analyzer R&S®TSMW is abbreviated as R&S TSMW.
R&S®MATLAB is abbreviated as R&S MATLAB .

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 intention 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 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.

Symbols and safety labels

⚠	⚠ 18 kg	⚡	♨	⏚	⏚	⏚	⚠
Notice, general danger location Observe product documentation	Caution when handling heavy equipment	Danger of electric shock	Warning! Hot surface	PE terminal	Ground	Ground terminal	Be careful when handling electrostatic sensitive devices

I ○	⏻	⎓	∼	≂	⧈
ON/OFF supply voltage	Standby indication	Direct current (DC)	Alternating current (AC)	Direct/alternating current (DC/AC)	Device fully protected by double (reinforced) insulation

Tags and their meaning

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

⚠ DANGER	indicates a hazardous situation which, if not avoided, will result in death or serious injury.
⚠ WARNING	indicates a hazardous situation which, if not avoided, could result in death or serious injury.
⚠ CAUTION	indicates a hazardous situation which, if not avoided, could result in minor or moderate injury.
NOTICE	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 tags 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 tags described here are always used only in connection with the related product documentation and the related product. The use of tags in connection with unrelated products or documentation can result in misinterpretation and in personal injury or material damage.

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, pollution severity 2, overvoltage category 2, use only indoors, max. operating altitude 2000 m above sea level, max. transport altitude 4500 m above sea level. A tolerance of ±10 % shall apply to the nominal voltage and ±5 % to the nominal frequency.
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 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 death.

Electrical safety

If the information on electrical safety is not observed either at all 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 an earthing contact and protective earth connection.
3. Intentionally breaking the protective earth connection 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 the product does not have a power switch for disconnection from the AC supply network, the plug of the connecting cable is regarded as the disconnecting device. In such cases, always ensure that the power plug is easily reachable and accessible at all times (corresponding to the length of connecting cable, approx. 2 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, a disconnecting device must be provided at the system level.
5. Never use the product if the power cable is damaged. Check the power cable on a regular basis to ensure that it is in proper operating condition. By taking appropriate safety measures and carefully laying the power cable, you can ensure that the cable will not be damaged and that no one can be hurt by, for example, tripping over the cable or suffering an electric shock.
6. The product may be operated only from TN/TT supply networks fused 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. 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 $V_{rms} > 30$ V, suitable measures (e.g. appropriate measuring equipment, fusing, 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 PE terminal on site and the product's PE 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 fused 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.
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", 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. 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).

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, PE conductor test, insulation resistance measurement, leakage current measurement, functional test). This helps ensure the continued safety of the product.

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. Keep cells and batteries out of the hands of children. If a cell or a battery has been swallowed, seek medical aid immediately.
5. Cells and batteries must not be exposed to any mechanical shocks that are stronger than permitted.
6. 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.
7. 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.
8. 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

1. 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.
2. 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.

Informaciones elementales de seguridad

Es imprescindible leer y observar 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.

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.

Símbolos y definiciones de seguridad

⚠	⚠ 18 kg	⚡	♨	⏚	⏚	⊥	⚠
Aviso: punto de peligro general Observar la documentación del producto	Atención en el manejo de dispositivos de peso elevado	Peligro de choque eléctrico	Adver-tencia: superficie caliente	Conexión a conductor de protección	Conexión a tierra	Conexión a masa	Aviso: Cuidado en el manejo de dispositivos sensibles a la electrostática (ESD)

| ○	⏻	⎓	∼	≂	⧈
Tensión de alimentación de PUESTA EN MARCHA / PARADA	Indicación de estado de espera (Standby)	Corriente continua (DC)	Corriente alterna (AC)	Corriente continua / Corriente alterna (DC/AC)	El aparato está protegido en su totalidad por un aislamiento doble (reforzado)

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	PELIGRO identifica un peligro inminente con riesgo elevado que provocará muerte o lesiones graves si no se evita.
⚠ ADVERTENCIA	ADVERTENCIA identifica un posible peligro con riesgo medio de provocar muerte o lesiones (graves) si no se evita.
⚠ ATENCIÓN	ATENCIÓN identifica un peligro con riesgo reducido de provocar lesiones leves o moderadas si no se evita.
AVISO	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.

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, grado de suciedad 2, categoría de sobrecarga eléctrica 2, 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 ±10 % sobre el voltaje nominal y de ±5 % sobre la frecuencia nominal.
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, pueden causarse lesiones o 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, se deberá considerar el enchufe del cable de conexión como interruptor. En estos casos se deberá asegurar que el enchufe siempre sea de fácil acceso (de acuerdo con la longitud del cable de conexión, aproximadamente 2 m). Los interruptores de función o electrónicos no son aptos para el corte de la red eléctrica. Si los productos sin interruptor están integrados 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.
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 $U_{eff} > 30$ 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.

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", 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 señalizar 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. 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).

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.
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. Mantener baterías y celdas fuera del alcance de los niños. En caso de ingestión de una celda o batería, avisar inmediatamente a un médico.
5. Las celdas o baterías no deben someterse a impactos mecánicos fuertes indebidos.

6. 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.
7. 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).
8. 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.
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

1. Si se trabaja de manera mecánica y/o térmica cualquier producto o componente más allá del funcionamiento previsto, pueden liberarse sustancias peligrosas (polvos 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.
2. 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.

Certified Quality System

DIN EN ISO 9001 : 2000
DIN EN 9100 : 2003
DIN EN ISO 14001 : 2004

DQS REG. NO 001954 QM UM

QUALITÄTSZERTIFIKAT

Sehr geehrter Kunde,
Sie haben sich für den Kauf eines Rohde & Schwarz-Produktes entschieden. Hiermit erhalten Sie ein nach modernsten Fertigungsmethoden hergestelltes Produkt. Es wurde nach den Regeln unseres Managementsystems entwickelt, gefertigt und geprüft.
Das Rohde & Schwarz Managementsystem ist zertifiziert nach:

DIN EN ISO 9001:2000
DIN EN 9100:2003
DIN EN ISO 14001:2004

CERTIFICATE OF QUALITY

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:

DIN EN ISO 9001:2000
DIN EN 9100:2003
DIN EN ISO 14001:2004

CERTIFICAT DE QUALITÉ

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 conformément aux normes:

DIN EN ISO 9001:2000
DIN EN 9100:2003
DIN EN ISO 14001:2004

ROHDE&SCHWARZ

PD 5213.8744.99 · V 01.00 · May 2007

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.

USA & Canada

Monday to Friday	(except US public holidays)
8:00 AM – 8:00 PM	Eastern Standard Time (EST)
Tel. from USA	888-test-rsa (888-837-8772) (opt 2)
From outside USA	+1 410 910 7800 (opt 2)
Fax	+1 410 910 7801
E-mail	CustomerSupport@rohde-schwarz.com

East Asia

Monday to Friday	(except Singaporean public holidays)
8:30 AM – 6:00 PM	Singapore Time (SGT)
Tel.	+65 6 513 0488
Fax	+65 6 846 1090
E-mail	CustomerSupport@rohde-schwarz.com

Rest of the World

Monday to Friday	(except German public holidays)
08:00 – 17:00	Central European Time (CET)
Tel. from Europe	+49 (0) 180 512 42 42*
From outside Europe	+49 89 4129 13776
Fax	+49 (0) 89 41 29 637 78
E-mail	CustomerSupport@rohde-schwarz.com

* 0.14 €/Min within the German fixed-line telephone network, varying prices for the mobile telephone network and in different countries.

ROHDE&SCHWARZ

Address List

Headquarters, Plants and Subsidiaries

Headquarters

ROHDE&SCHWARZ GmbH & Co. KG Mühldorfstraße 15 · D-81671 München P.O.Box 80 14 69 · D-81614 München	Phone +49 (89) 41 29-0 Fax +49 (89) 41 29-121 64 info.rs@rohde-schwarz.com

Plants

ROHDE&SCHWARZ Messgerätebau GmbH Riedbachstraße 58 · D-87700 Memmingen P.O.Box 16 52 · D-87686 Memmingen	Phone +49 (83 31) 1 08-0 +49 (83 31) 1 08-1124 info.rsmb@rohde-schwarz.com
ROHDE&SCHWARZ GmbH & Co. KG Werk Teisnach Kaikenrieder Straße 27 · D-94244 Teisnach P.O.Box 11 49 · D-94240 Teisnach	Phone +49 (99 23) 8 50-0 Fax +49 (99 23) 8 50-174 info.rsdts@rohde-schwarz.com
ROHDE&SCHWARZ závod Vimperk, s.r.o. Location Spidrova 49 CZ-38501 Vimperk	Phone +420 (388) 45 21 09 Fax +420 (388) 45 21 13
ROHDE&SCHWARZ GmbH & Co. KG Dienstleistungszentrum Köln Graf-Zeppelin-Straße 18 · D-51147 Köln P.O.Box 98 02 60 · D-51130 Köln	Phone +49 (22 03) 49-0 Fax +49 (22 03) 49 51-229 info.rsdc@rohde-schwarz.com service.rsdc@rohde-schwarz.com

Subsidiaries

R&S BICK Mobilfunk GmbH Fritz-Hahne-Str. 7 · D-31848 Bad Münder P.O.Box 20 02 · D-31844 Bad Münder	Phone +49 (50 42) 9 98-0 Fax +49 (50 42) 9 98-105 info.bick@rohde-schwarz.com
ROHDE&SCHWARZ FTK GmbH Wendenschloßstraße 168, Haus 28 D-12557 Berlin	Phone +49 (30) 658 91-122 Fax +49 (30) 655 50-221 info.ftk@rohde-schwarz.com
ROHDE&SCHWARZ SIT GmbH Am Studio 3 D-12489 Berlin	Phone +49 (30) 658 84-0 Fax +49 (30) 658 84-183 info.sit@rohde-schwarz.com
R&S Systems GmbH Graf-Zeppelin-Straße 18 D-51147 Köln	Phone +49 (22 03) 49-5 23 25 Fax +49 (22 03) 49-5 23 36 info.rssys@rohde-schwarz.com
GEDIS GmbH Sophienblatt 100 D-24114 Kiel	Phone +49 (431) 600 51-0 Fax +49 (431) 600 51-11 sales@gedis-online.de
HAMEG Instruments GmbH Industriestraße 6 D-63533 Mainhausen	Phone +49 (61 82) 800-0 Fax +49 (61 82) 800-100 info@hameg.de

Locations Worldwide

Please refer to our homepage: www.rohde-schwarz.com

- Sales Locations
- Service Locations
- National Websites

Table of Contents

Documentation Overview

The user documentation for the R&S TSMW Interface is divided as follows:

Conventions Used in the Documentation

The following conventions are used throughout the R&S TSMW Interface Programming 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, softkeys, menus, options, buttons etc., are enclosed by quotation marks.
“KEYS”	Key names are written in capital letters and enclosed by quotation marks.
Input	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.

1 Introduction

This manual describes all available programming interfaces for R&S TSMW.

The R&S TSMW is a high-power platform for optimizing all conventional mobile radio networks. Two highly sensitive 20 MHz frontends for center frequencies from 30 MHz to 6 GHz, a dual-channel preselector and an FPGA-based software-defined architecture offer unsurpassed performance while providing maximum flexibility and future-proofness. In addition to functioning as a mobile radio scanner, the R&S TSMW makes also an ideal digital I/Q receiver for customer specific applications.

In chapter R&S TSMW signal processing block diagram on p. 15 you can find a block diagram of R&S TSMW.

Related Documentation

- For further information about the R&S TSMW and its functionality see the Operation Manual of R&S TSMW Radio Network Analyzer.
- For information of the general MATLAB functions see the online help in MATLAB.

2 R&S TSMW-K1

An integral part of the R&S TSMW is its R&S TSMW-K1 application. The application provides a flexible MATLAB interface as well as an equivalent C++ function interface for performing measurements directly on the R&S TSMW and processing the results on the PC. This enables you, for example, not only to design and analyze receiver algorithms in MATLAB, but also to port them to C++ for a real time version. Or you can even perform technology-independent channel measurements, which can be used to simulate realistic fading scenarios in the lab.

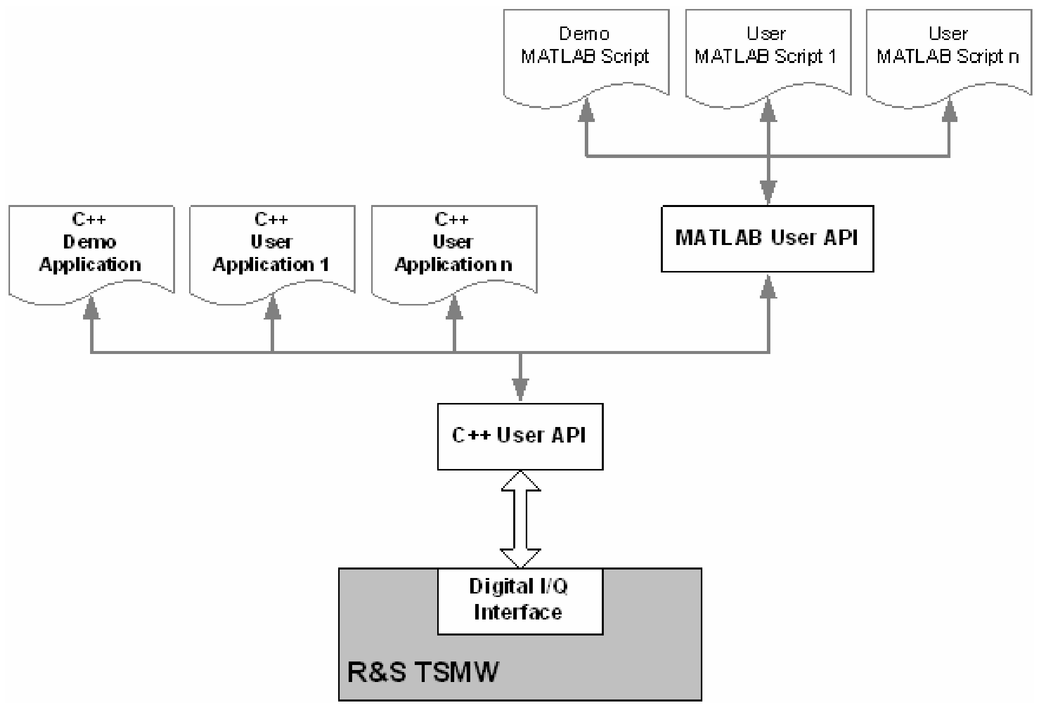


Figure 2-1: Digital I/Q interface overview

Several MATLAB and C++ example scripts are available on CD-ROM and on installation folder `<TSMW-K1 installation directory>\Examples`. They demonstrate example implementations of R&S TSMW-K1 functionalities.

2.1 What's new

- Support of periodic measurement
  - New MATLAB functions. See Appendix M-Functions on page 57.
  - New MATLAB structure. See chapter Structures on page 23
  - New C++ functions. See Appendix C++ Header Files on page 57.
- Resource management for concurrent measurement requests
  - New MATLAB functions. See Appendix M-Functions on page 57.
  - New MATLAB structure. See chapter Structures on page 23
  - New C++ functions. See Appendix C++ Header Files on page 57
- Enhanced trigger support
  - New MATLAB functions. See Appendix M-Functions on page 57.
  - New MATLAB structure. See chapter Structures on page 23
  - New C++ functions. See Appendix C++ Header Files on page 57.
- Enhanced GPS support
  - New C++ functions. See Appendix C++ Header Files on page 57.
- Improved manual structure.
  For faster finding information about the available TSMW-K1 functions and parameters in MATLAB and C++ the API documentation is assembled within the appendix:
  - API documentation for MATLAB see Appendix M-Functions on page 57.
  - API documentation for C++ see Appendix C++ Header Files on page 57.

For more details about new features, see the Release Notes of the current R&S TSMW-K1 software version.

2.2 System Requirements

- 2GB RAM
- Gigabit Ethernet Adapter supporting 9kb Jumbo Frames
- CD-ROM drive (for installation only)
- MS Windows XP, Service Pack 2
- R&S TSMW
- MATLAB Software (including Filter Design Toolbox) Version R2007b (mandatory when using MATLAB source files, not necessary for compiled MATLAB demo application or when using C++ API)

2.2.1 Additional Requirements for I/Q Streaming

To use the R&S TSMW feature "I/Q streaming" additional requirements on the computer equipment is essential.

- PCI-Express Gigabit Ethernet Adapter
- High-speed hard disk with >40MB/s sustained transfer rate (write) (for calculating the actually necessary disc speed see equation below)

This is caused by the large amount of data created during I/Q streaming.

Based on above facts it is useful to calculate the I/Q data rate and to define the minimum system requirements for the computer equipment.

Formula:

```
[Bitrate TSMW] = [Sample rate in MS/s]*2*(X + Y)*Z*OVal
```

X	Number of selected channels on RF 1.
Y	Number of selected channels on RF 2.
Z	Number of bits used for the data transfer. Possible values are 8 Bit, 12 Bit, 16 Bit and 20 Bit.
OVAL	Overhead value, approximately 1,1.

2.2.2 Example

Following parameters are set on R&S TSMW for a I/Q raw data streaming measurement:

`X = 1`	On RF1 two sub channels are selected.
`Y = 2`	On RF2 four sub channels are selected.
`Z = 20 Bit`	Data transfer format is 20 Bit.
`OVal = 1,1`	Overhead value 10%.
Sample rate `= 3[MS/s]`	Sample rate 3 [MS/s]

Supposed data transfer rate:

```
3000000 Samples/s * 2 * (1 + 2) * 20 Bit/Samples * 1,1
      = 396000000 Bit/s
      = 49,5 MByte/s
```

That mean during this I/Q data streaming measurement 49,5 MB per second has to be transferred from R&S TSMW to the computer over a Gigabit Ethernet card and saved on hard disk.

2.3 Install

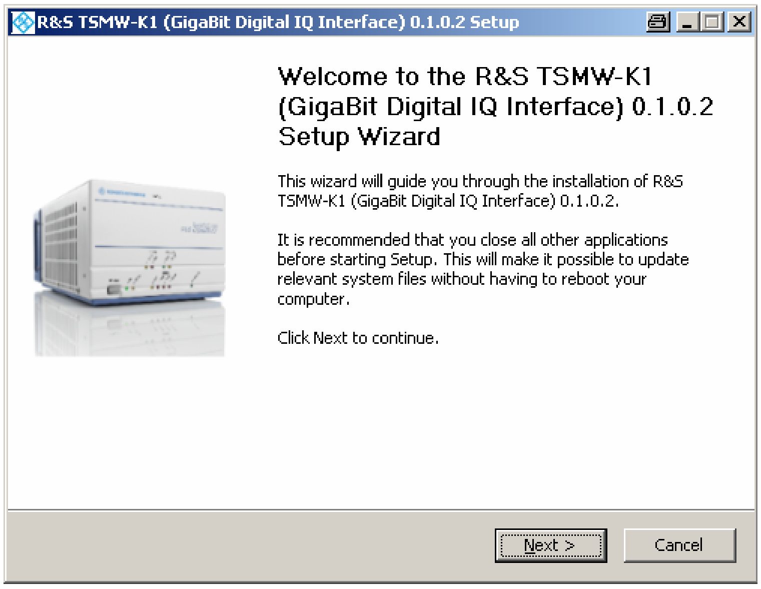
2.3.1 Installation of the R&S TSMW-K1 software

The installation of the software is controlled by Setup Wizard. If you have no full version of MATLAB R2007/R2008 installed, it is necessary to install the MATLAB Component Runtime (MCR). A compatible MCR version is on the CD-ROM.

Please keep in mind that the MCR allows only to run the compiled MATLAB Demo Application. Direct access to the corresponding R&S TSMW MATLAB functions is only possible with a full MATLAB installation.

Procedure:

1. Insert the R&S TSMW CD into the CD-ROM drive.
   The CD start menu opens automatically.
2. Select the menu item "Applications"-> "TSMW-K1 Setup".
   The "Setup Wizard" dialog starts immediately.



Welcome to the R&S TSMW-K1 (GigaBit Digital IQ Interface) 0.1.0.2 Setup Wizard

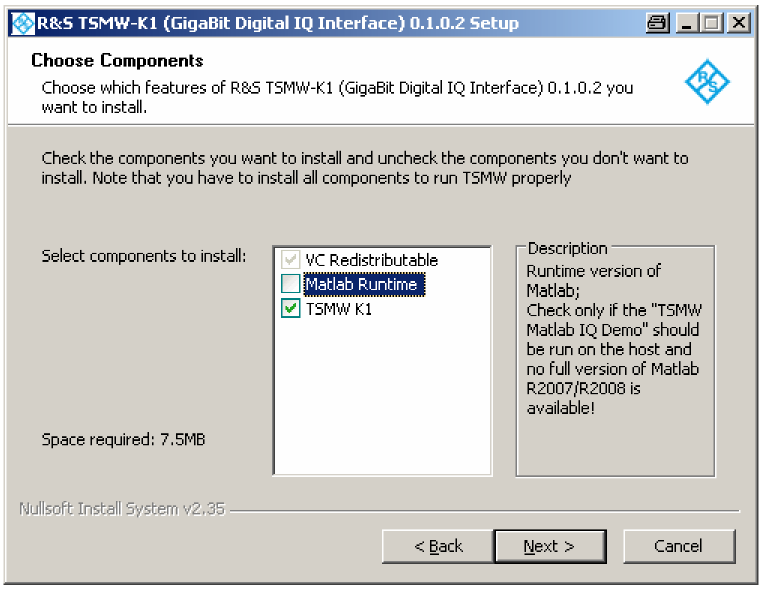
This wizard will guide you through the installation of R&S TSMW-K1 (GigaBit Digital IQ Interface) 0.1.0.2.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

Figure 2-2: Start dialog of the Setup Wizard

3. Click the “Next” button.
   The “License Agreement” window opens.
4. To continue the installation you have to agree with the End User License Agreement. Therefore click the “I Agree” button.
   The "Choose Components" window opens.



R&S TSMW-K1 (GigaBit Digital IQ Interface) 0.1.0.2 Setup

Choose Components

Choose which features of R&S TSMW-K1 (GigaBit Digital IQ Interface) 0.1.0.2 you want to install.

Check the components you want to install and uncheck the components you don't want to install. Note that you have to install all components to run TSMW properly

Select components to install:

- [✓] VC Redistributable
- [ ] Matlab Runtime
- [✓] TSMW K1

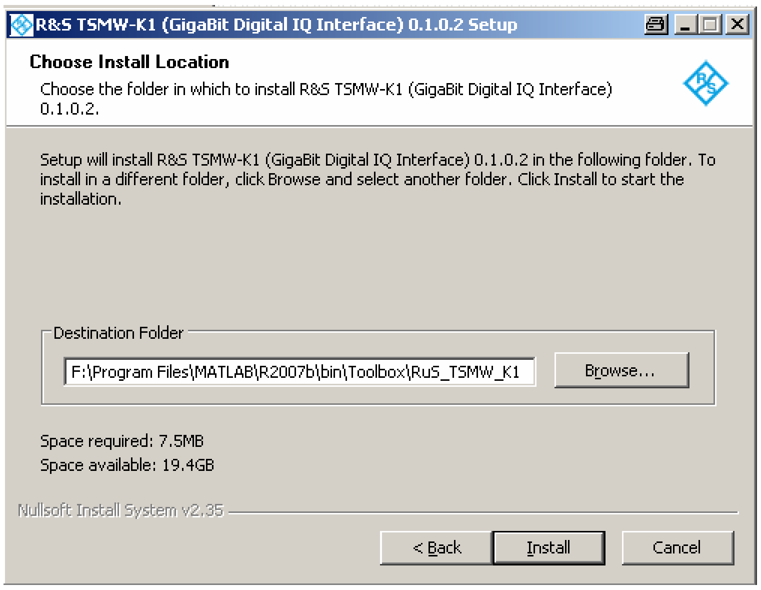
Description
Runtime version of Matlab;
Check only if the "TSMW Matlab IQ Demo" should be run on the host and no full version of Matlab R2007/R2008 is available!

Space required: 7.5MB

Nullsoft Install System v2.35

Figure 2-3: Choose Component window

5. Check the desired components to install:
   a. Check the “TSMW-K1“ component to install the R&S TSMW-K1 application. (Mandatory).
   b. Check the “Matlab Runtime” component if no full version of MATLAB R2007/R2008 is installed. (Optional)
6. Click the “Next” button to continue.
   The “Choose Install Location” window opens.



R&S TSMW-K1 (GigaBit Digital IQ Interface) 0.1.0.2 Setup

Choose Install Location

Choose the folder in which to install R&S TSMW-K1 (GigaBit Digital IQ Interface) 0.1.0.2.

Setup will install R&S TSMW-K1 (GigaBit Digital IQ Interface) 0.1.0.2 in the following folder. To install in a different folder, click Browse and select another folder. Click Install to start the installation.

Destination Folder
F:\Program Files\MATLAB\R2007b\bin\Toolbox\RuS_TSMW_K1

Space required: 7.5MB
Space available: 19.4GB

Nullsoft Install System v2.35

Figure 2-4: Choose Install Location window

7. It is recommended to install all files to the MATLAB toolbox folder. The exact path depends on the installation of MATLAB.
   I. e.: `C:\Program Files\MATLAB\R2007a\bin\toolbox\RuS_TSMW_K1`
   Click the “Install” button to start the installation.

8. The installation of the R&S TSMW-K1, VC Redistributable starts automatically.
9. During the installation a dialog window opens for the Redistributable Package. Follow the instruction and continue the installation.
10. If the component “Matlab Runtime” was selected the MATLAB Component Runtime (MCR) installer starts automatically.
    For detailed information how to install the MCR see Working with the MCR (http://www.mathworks.com/access/helpdesk/help/toolbox/compiler/f12-999353.html).
11. At the end of the installation process an installation complete window appears. Click the “Finish” button.

The installation is finished.

In the menu “Start”->”Programs”->”Rohde&Schwarz” exists a new menu folder for the R&S TSMW-K1 software with all tools and information about it.

2.3.2 Set the path in MATLAB to the R&S TSMW-K1 folder

It is required to add the installation path of the R&S TSMW-K1 to the MATLAB environment in order to have access to the R&S TSMW-K1 MATLAB functions.

Procedure:

1. Select in MATLAB on the menu bar "File" -> "Set Path…".
   The "Set Path" window opens.
2. Click the "Add Folder…" button.
   The "Browse for Folder" window opens.
3. Add the directory where the MATLAB wrapper functions are installed. I.e.:
   `C:\Program Files\MATLAB\R2007a\bin\toolbox\RuS_TSMW_K1\Matlab`
4. Add the directory where the R&S TSMW IQ Interface and related libraries are installed. I.e.:
   `C:\Program Files\MATLAB\R2007a\bin\toolbox\RuS_TSMW_K1\lib`
5. Click the "Save" button.

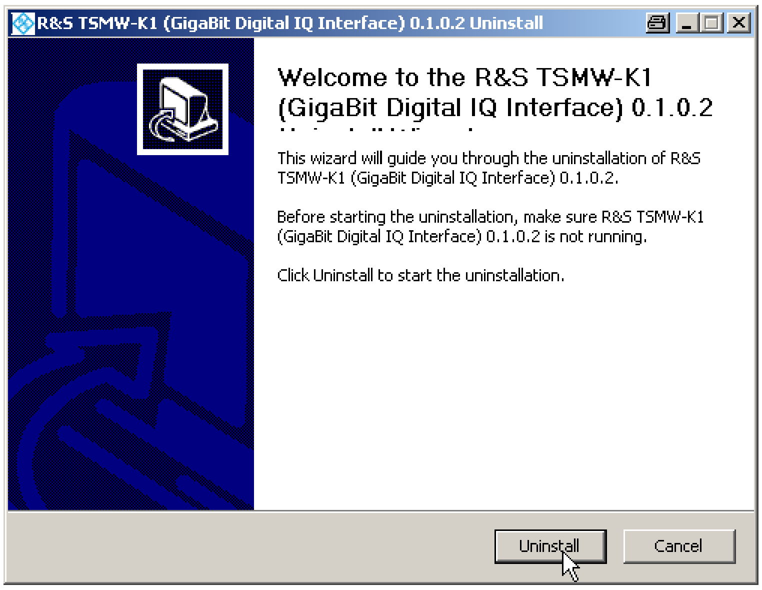
The R&S TSMW-K1 functions are now available in MATLAB.

2.4 Uninstall

The R&S TSMW-K1 application offers a wizard to guide you through the uninstall process of the software.

Procedure:

1. Select "Start"->"Programs"->"Rohde&Schwarz"->"TSMW-K1 <*ver. no.*>"->"Uninstall <*ver. no.*>".
   The uninstall wizard window opens.



R&S TSMW-K1 (GigaBit Digital IQ Interface) 0.1.0.2 Uninstall

Welcome to the R&S TSMW-K1 (GigaBit Digital IQ Interface) 0.1.0.2

This wizard will guide you through the uninstallation of R&S TSMW-K1 (GigaBit Digital IQ Interface) 0.1.0.2.

Before starting the uninstallation, make sure R&S TSMW-K1 (GigaBit Digital IQ Interface) 0.1.0.2 is not running.

Click Uninstall to start the uninstallation.

Figure 2-5: Uninstall wizard

2. Follow the introduction of the wizard to uninstall the application.

If no error occurs the application is successfully uninstalled.

2.5 R&S TSMW signal processing block diagram

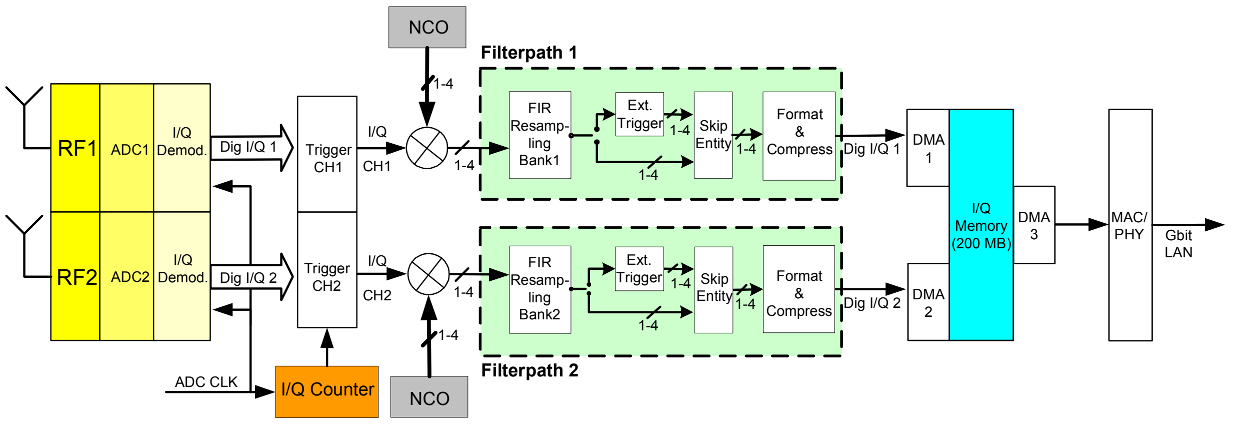


Figure 2-6: R&S TSMW signal processing block diagram

`RF1`
`RF2`
	Control the frequency setting for frontend 1 and 2.
	`RF1` and `RF2` are configurable via the following m-structure fields:
	`MeasCtrl.MeasCtrl.AttStrategy,`
	`MeasCtrl.MeasCtrl.Splitter,`
	`MeasCtrl.ChannelCtrl1.Frequency,`
	`MeasCtrl.ChannelCtrl2.Frequency,`
	`MeasCtrl.ChannelCtrl1.Attenuation,`
	`MeasCtrl.ChannelCtrl2.Attenuation,`
	`MeasCtrl.ChannelCtrl1.Preamp,`
	`MeasCtrl.ChannelCtrl2.Preamp,`
	`MeasCtrl.ChannelCtrl1.CalibInput,`
	`MeasCtrl.ChannelCtrl2.CalibInput,`
	`TSMWOptions.Frontends, TSMWOptions.AMPS_CH1,`
	`TSMWOptions.AMPS_CH2, TSMWOptions.Mode`
	For further information about the fields refer to chapter Structure: MeasCtrl on page 24, Structure: TSMWOptionson page 33.

`I/Q Demodulation 1`
`I/Q Demodulation 2`
	Controls the RF signal processing technique.

`I/Q CLK`	Native R&S TSMW I/Q sample clock of `21.9444MHz`.
`I/Q Counter`	`I/Q Counter` counts the I/Q samples starting at the time when a connection to the R&S TSMW has been established.

`Trigger CH1`
`Trigger CH2`
	Controls the start time of a measurement for channel 1 and 2. The specified I/Q start time is the same for all sub channels of channel 1 or 2, respectively. Measurements on different frontends can be started either independently or simultaneously.
	When using a resampling-filter, automatically the measurement is started `GroupDelay` samples before the specified I/Q sample time. `GroupDelay` is a parameter of the filter specification.
	`Trigger CH1` and `CH2` are configurable via parameter `StartTimes` in m-function `TSMWIQMeasure`.

`NCO`	Numerical Control Oscillator. Controls the frequency shift for the 1 to 4 sub channels for frontend 1 and 2. The following FIR Resampling Bank will perform low pass filtering and fractional resampling for each sub channel using the user defined resampling filter.

	Note:
	The filter and resampling factor is the same for all sub channels but can be different for each frontend.
	
	`NCO` is configurable via the following m-structure fields:
	`MeasCtrl.ChannelCtrl1.NoOfChannels,`
	`MeasCtrl.ChannelCtrl2.NoOfChannels,`
	`MeasCtrl.ChannelCtrl1.FreqShift[],`
	`MeasCtrl.ChannelCtrl2.FreqShift[].`
	For further information about the fields refer to chapter Structure: MeasCtrl on page 24.

`FIR Resampling Bank 1`
`FIR Resampling Bank 2`
	Controls the used filter type for frontend 1 and 2.
	Note:
	The corresponding filter specification has to be transferred to the R&S TSMW via the m-function `TSMWIQSetup`.
	A filter specification has the following m-structure fields:
	`FilterSpec.FilterID, FilterSpec.NoOfCoeffs, FilterSpec.OversplFact, FilterSpec.FilterCorrdB, ResultShiftNumber, FilterSpec.NDown`
	Which filter is active on `FIR Resampling Bank 1` and `2` are configurable via the m-structure fields:
	`MeasCtrl.MeasCtrl.FilterType,`
	`MeasCtrl.MeasCtrl.FilterID.`
	For further information about the fields refer to chapter Structure: MeasCtrl on page 24 and Structure: FilterSpec on page 32.

`Ext. Trigger 1`
`Ext. Trigger 2`
	The external trigger units activate or deactivate the I/Q data measurement when the positive or negative edge of the trigger signal was reached.
	`External Trigger 1` and `2` are configurable via m-function TSMWTrigger.
	For `Trigger 1` and `2` own m-structure fields exist:
	`MeasCtrl.TriggerCtrl.Cmd,`
	`MeasCtrl.TriggerCtrl.Mode,`
	`MeasCtrl.TriggerCtrl.Falling,`
	`MeasCtrl.TriggerCtrl.TriggerLine,`
	`MeasCtrl.TriggerCtrl.MeasRequestID,`
	`MeasCtrl.TriggerCtrl.Att[2],`
	`MeasCtrl.TriggerCtrl.Preamp[2].`
	For further information about the fields refer to chapter Structure: MeasCtrl on page 24.

`Skip Entity 1`
`Skip Entity 2`
	Allow to skip a variable number of samples for each sub channel after the measurement on the corresponding frontend has started before I/Q samples are recorded on this sub channel.
	`Skip Entity 1` and `2` are configurable via the following m-structure fields:
	`MeasCtrl.ChannelCtrl1.ChannelDelay[],`
	`MeasCtrl.ChannelCtrl2.ChannelDelay[].`
	For further information about the fields refer to chapter Structure: MeasCtrl on page 24.

`Format & Compress`	Controls the I/Q data compression. I/Q data are converted into a block floating point format. 64 I/Q samples get always the same exponent. This

	allows an optimum usage of the I/Q bit width during transmission to the host PC. The I/Q data width (without exponent) used for transmission to the host PC is specified by `MeasCtrl.MeasCtrl.DataFormat`. For further information about the field refer to chapter Structure: MeasCtrl on page 24.
`DMA 1, DMA 2, DMA 3`	Direct Memory Access
`I/Q Memory`	I/Q data memory.
`MAC/Phy`	Specifies the MAC-Address and the Physical-Address of the R&S TSMW.

Table 2-1: : Correlation between R&S TSMW signal processing and m-structures, m-functions.

2.6 Features

2.6.1 Measurement Types

The R&S TSMW-K1 option supports different I/Q measurement types:

Types	**Description**
Single measurement	Performs a single I/Q data measurement. **Periodic measurement** Optionally single measurements could be performed as periodic measurements. A periodic measurement performs in equal intervals a single measurement. When the fields of periodic measurement in m-structure `MeasCtrl` exist., a single measurement is automatically a periodic measurement. For further information about these fields refer to chapter MeasCtrl values for trigger control on page 29. For further information see the short description of the m-functions Measurement on page 20 and the demo. application on page 36.
Stream measurement	Performs continuous measurements and write the measured I/Q data into one file until the maximum size is reached. Two modes are distinguish: **"Online" streaming** The stream data is processed online. Hence stream data is only buffered in memory (up to the given stream buffer size) until it is acquired via one of the TSMWIQGetStreamXXX functions. As soon as the corresponding data has been acquired, the corresponding memory is automatically released. **"Offline" streaming** The data is automatically written to a stream data file. After the streaming has finished, the stream data file can be opened and read. For more details refer to the m-functions Seamless streaming of I/Q data in real time on page 21 and see the description of the demo. application on page 36.

2.6.2 Resource management for concurrent measurement requests

The R&S TSMW-K1 supports a scalable resource management for I/Q data measurement tasks.

The basic idea is that concurrent measurement requests can be assigned different portions of the available receiver time. This becomes in particular important for applications where independent and concurrent measurement tasks perform measurements on the R&S TSMW.

In order to deal with this problem, it is possible to allocate receiver resources (i.e. receiver time for frontend 1 or 2) and assign measurements to these resources. Having done this, the R&S TSMW internal measurement scheduler takes the current usage of each acquired receiver usage into account when deciding whether a measurement shall be performed or not.

Using resource management does only make sense if concurrent measurement requests exist. This means situations appear where a large number of measurement requests are started that cannot be served at a time but may have to be delayed. If start times are specified for these measurement requests, it makes sense (if possible) to specify several, alternative start times in order to allow the scheduler to start the corresponding measurement at a later time. If a start time cannot be met and no alternative start times are available, the measurement request is canceled.

Example:

Three receivers resourced as follows:

- Resource 1: 30% of frontend 1
- Resource 2: 50% of frontend 2
- Resource 3: 70% of frontend 1 and 50% of frontend 2

This is done with the resource request functions, see chapter Measurement on p. 20.

When a measurement is started, the resource request fields within the MATLAB structure `MeasCtrl` are used to assign each measurement that is started to the right resource by specifying the corresponding resource ID. For details see chapter MeasCtrl values for capacity request for RF channels on p. 26).

The R&S TSMW scheduler takes the usage of the specified resource into account when deciding which of the concurrent measurement requests to serve first.

Schedule algorithm for measurement tasks

If too many competing measurement tasks request the same resource, the system schedules all waiting measurement tasks automatically. The schedule algorithm manages the access to the resource according to the priority number (m-structure field `MeasCtrl.MeasCtrl.Priority`). It is possible to give the measurement task a priority value between 0 and 15. 15 is the priority with the highest preferences for resource access. The given priority numbers increase dynamically according to the waiting time (observation time).

If a measurement task does not get at a priority value at creation time, the system will attach the lowest possible priority value to it => 0.

2.6.3 Trigger

R&S TSMW supports up to two external trigger sources. The R&S TSMW-K1 API allows configuring positive/negative edge of the trigger signal for measurements.

2.6.4 GPS

An integrated SuperSense GPS receiver with 16 channels and a refresh rate of 4 Hz also allows the R&S TSMW to be used in areas with weak GPS signals. The GPS receiver sends data in NMEA and UBX format. Therefore it is possible to send control commands for NMEA and UBX to the R&S. This is possible via the MATLAB and C++ functions of R&S TSMW-K1 API.

2.7 Function Overview

Each function returns an error code: 0 if successful. Otherwise a number is returned which defines the error code. To get the corresponding error text calls the function TSMWGetLastError. The function returns the error information of the occurred error and reset the error code variable again to zero.

Variable ErrorCode

Please keep in mind that only one ErrorCode variable is used for all functions. Therefore only the current returned error code is available.

2.7.1 General

The following functions allow to control and access the I/Q interface of the R&S TSMW. Furthermore trigger settings and error handling is managed.

Function MATLAB (*.m file)	Function C++ (header files)	Description
`TSMWInitInterface`	`TSMWInitInterface_c`	Load and initialize the R&S TSMW IQ interface library `TSMWIQInterface.dll`.
`TSMWReleaseInterface`	`TSMWReleaseInterface_c`	Unload the R&S TSMW MATLAB interface library `TSMWIQInterface.dll` and disconnect from R&S TSMW(s).
`TSMWConnect`	`TSMWConnect_c`	Establish a connection to the R&S TSMW with the given IP address and given options.
`TSMWGetLastError`	`TSMWGetLastError_c`	Return the error message of the occurred error.
`TSMWShutdown`	`TSMWShutdown_c`	Shutdown the R&S TSMW.

Function MATLAB (*.m file)	Function C++ (header files)	Description
TSMWGetIQTime	TSMWGetIQTime_c	Get current I/Q time of given R&S TSMW.
TSMWTrigger	TSMWTrigger_c	Set/get the R&S TSMW trigger output/ input.
TSMWGetVersion	TSMWGetVersion_c	Get R&S TSMW I/Q interface version.
TSMWGetVersionText	TSMWGetVersionText_c	Get R&S TSMW I/Q interface version as text.

Table 2-2: Overview of the MATLAB and C++ I/Q interface functions.

2.7.2 Measurement

These functions configure, setup and control measurements. Furthermore it is possible to manage the scalable resource of the two RF to the completive measurement tasks und organize the transfer of the I/Q data to the pc for further handling.

2.7.2.1 Functions

Function MATLAB (*.m file)	Function C++ (header files)	Description
TSMWIQSetup	TSMWIQSetup_c	Send a filter specification to the R&S TSMW.
TSMWIQMeasure	TSMWIQMeasure_c	Start a measurement with the given parameters.
	TSMWIQMeasureExt_c	Start an extended measurement.
	TSMWIQMeasureTrig_c	Start a triggered measurement.
-	TSMWIQTriggerCtrl_c	Change parameters of a triggered measurement.
TSMWResourceRequest	TSMWIQResourceRequest_c	Request receiver resources.
TSMWIQPeriodCtrl	TSMWIQPeriodCtrl_c	Change parameters of a running periodic measurement request.
TSMWIQDataAvailable	TSMWIQDataAvailable_c	Check how many measurement results from R&S TSMW are available.
TSMWGetIQResultParam	TSMWGetIQResultParam_c	Get measurement result parameters.
TSMWIQGetData	TSMWIQGetDataInt16_c	Retrieves I/Q data as complex double array.
TSMWIQGetDataInt16	TSMWIQGetDataInt16_c	Return the measurement data in 16 bit integer array format.
TSMWIQGetDataInt32	TSMWIQGetDataInt32_c	Return the measurement data in 32 bit integer array format.

`TSMWIQGetDataSingle`	`TSMWIQGetDataSingle_c`	Return the measurement data in single precision floating point array format.
`TSMWIQGetDataDouble`	`TSMWIQGetDataDouble_c`	Return the measurement data in double precision floating point array format.
`TSMWIQGetDataSingleIlv`	`TSMWIQGetDataSingleIlv_c`	Return the measurement data in interleave single precision floating point array format.
`TSMWIQGetDataDoubleIlv`	`TSMWIQGetDataDoubleIlv_c`	Return the measurement data in interleave double precision floating point array format.
`TSWMGetFIRParam`	-	Calculate FIR parameters for a given downsampling factor.

Table 2-3: Overview of the MATLAB and C++ I/Q interface functions.

2.7.2.2 Templates

Function MATLAB (*.m file)	**Function C++ (header files)**	**Description**
`MeasCtrlTemplate`	-	Create a template for the MeasCtrl structure for double channel (MIMO) measurements with default values.
`MeasCtrlTemplate_RF1`	-	Create a template for the MeasCtrl structure for RF channel 1 measurements.
`MeasCtrlTemplate_RF2`	-	Create a template for the MeasCtrl structure for RF channel 2 measurements.

Table 2-4: Overview of the MATLAB and C++ I/Q interface functions.

2.7.3 Seamless streaming of I/Q data in real time

The measured I/Q data is transmitted in real time and available on PC. Both "online" and "offline" streaming is possible. Streaming can in general be started with the same parameters as normal (block) measurements. However, bandwidth (data rate) limitations have to be considered for successful streaming without lost data blocks.

R&S TSWM allows to perform technology-independent channel measurements. These measurements can be used to simulate realistic fading scenarios in a laboratory environment. To do this, I/Q data recorded by the R&S TSMW can, for example, be replayed directly on a signal generator from Rohde & Schwarz. This data, which has been recorded under real conditions, can therefore be replayed again and again in the laboratory environment. The seamless streaming of I/Q data can be carried out with a maximum bandwidth of 20 MHz. The duration of an I/Q recording is determined solely by the size of the hard disk.

2.7.3.1 Functions

Function MATLAB **(*.m file)**	**Function C++** **(header files)**	**Description**
TSMWIQStream	TSMWIQStream_c	Start streaming measurement for a given stream ID.
TSMWTGEnable	TSMWTGEnable_c	Enable the tracking generator (test) output.
TSMWSetDAC	TSMWSetDAC_c	Set the reference oscillator DAC value.
TSMWIQStreamStatus	TSMWIQStreamStatus_c	Return the I/Q streaming status.
TSMWIQStopStreaming	TSMWIQStopStreaming_c	Stop streaming for a given stream ID.
TSMWIQOpenStreamFile	TSMWIQOpenStreamFile_c	Open the streaming file.
TSMWIQCloseStreamFile	TSMWIQCloseStreamFile_c	Close the streaming file.
TSMWIQGetStreamSingle	TSMWIQGetStreamSingle_c	Return the streaming data block in single format.
TSMWIQGetStreamDouble	TSMWIQGetStreamDouble_c	Return the streaming data block in double format.
TSMWIQGetStreamSingleIlv	TSMWIQGetStreamSingleIlv_c	Return the streaming data block in interleave single format.
TSMWIQGetStreamDoubleIlv	TSMWIQGetStreamDoubleIlv_c	Return the streaming data block in interleave double format.

Table 2-5: Overview of the MATLAB and C++ I/Q interface functions.

2.7.3.2 Templates

Function MATLAB **(*.m file)**	**Function C++** **(header files)**	**Description**
`MeasCtrlTemplate_RF1Stream`	-	Create a template for the MeasCtrl structure for RF channel 1 streaming measurement.
`MeasCtrlTemplate_RF2Stream`	-	Create a template for the MeasCtrl structure for RF channel 2 streaming measurement.

Table 2-6: Overview of the MATLAB and C++ I/Q interface functions.

2.7.4 GPS

These functions allows to configure and receive GPS data from R&S TSMW. NMEA and UBX messages are supported.

Function MATLAB (*.m file)	**Function C++** (header files)	**Description**
`TSMWGPSSync`	`TSMWGPSSync_c`	Enable or disable the GPS synchronization.
	`TSMWGPSClearBuffer_c`	Clear the GPS NMEA buffer.
`TSMWGetIQTime`	`TSMWGetIQTime_c`	Get current IQ time of given R&S TSMW.
`TSMWGPSEnable`	`TSMWGPSEnable_c`	Enable or disable GPS data subscription from R&S TSMW.
`TSMWGPSGetNMEALine`	`TSMWGPSGetNMEALine_c`	Get GPS NMEA text line.
`TSMWGPSSendNMEACmd`	`TSMWGPSSendNMEACmd_c`	Send NMEA command string to GPS.
`TSMWExtRefEnable`	`TSMWExtRefEnable_c`	Enable external reference.
-	`TSMWGPSSendUBXCmd_c`	Send UBX message to GPS.
-	`TSMWGPSGetUBXMsg_c`	Get next available GPS UBX message

Table 2-7: Overview of the MATLAB and C++ I/Q interface functions.

2.8 Structures

For easier working with the R&S TSMW MATLAB I/Q interface API, several parameter structures are defined. The following sub chapters explain the structures with their fields. Additionally the corresponding C++ structure is mention. Most C++ parameters correspond to the available MATLAB fields.

Example of MATLAB structures

Examples are available in the example folder

`<TSMW-K1 installation directory>\Examples\Matlab.`

2.8.1 Structure: MeasCtrl

Description:

The structure contains measurement control values. It is divisible into the following parts:

- **General measurement control field**
  Defines general measurement control values.
- **Stream control field**
  Defines the streaming options.
- **Resource request field**
  Defines the requested bandwidth of the RF.
- **Schedule control field**
  Defines the schedule options.
- **Period control field**
  Defines the settings for periodic measurement.
- **Trigger control field**
  Defines the trigger options.
- **RF channel 1 control field**
  Defines specific RF channel 1 control values.
- **RF channel 2 control field**
  Defines specific RF channel 2 control values.

For easier handling in MATLAB these four components are put together in one structure `MeasCtrl`.

In C++ these measurement control structures are defined as structure types (`typdef struct`):

- General measurement values
  `struct TSMW_IQIF_MEAS_CTRL`
- Stream control
  `struct TSMW_IQIF_STREAM_CTRL`
- Independent RF channel control
  `struct TSMW_IQIF_CH_CTRL`
- Configure resource request
  `struct TSMW_IQIF_RESOURCE_PARAM`
- Schedule control
  `struct TSMW_IQIF_SCHEDULE_CTRL`
- Periodic measurement control
  `struct TSMW_IQIF_PERIOD_CTRL`
- Trigger control
  `struct TSMW_IQIF_TRIG_CTRL`

2.8.1.1 MeasCtrl values for general measurement control

Description:

These fields allow configuring the general measurement control settings on the R&S TSMW. For faultless use the values has to be set and submit over the Digital I/Q interface to the TSMW(s).

Field(s):

`MeasCtrl.NoOfSamples`	Specifies the number of I/Q samples to measure.
`MeasCtrl.FilterType`	Specifies the used filter type: 0: Use predefined filters 1: Use user defined filters. (They have to be sent to the R&S TSMW before).
`MeasCtrl.FilterID`	If a predefined filter is used this field define the filter number.
`MeasCtrl.DataFormat`	Specifies I/Q data compression format for block wise compression. 0: 2x 8 Bit per complex sample 1: 2x 12 Bit per complex sample 2: 2x 16 Bit per complex sample 3: 2x 20 Bit per complex sample
`MeasCtrl.AttStrategy`	Specifies the attenuation strategy. 0: Manual attenuation. 1: Auto attenuation. If activated and the signal strength is to strong: 1. The preamplifier is deactivated. 2. If the signal strength is still to strong, the attenuation will be increased in 5dB steps till the signal strength is ok. **Note:** After each reconfiguration step the I/Q measurement is restarted. **Note:** Only available for block measurements.
`MeasCtrl.Splitter`	Specifies the RF channel 1 splitter. Splitting the signal after preselector to both frontends. 0: Disable splitter 1: Enable splitter
`MeasCtrl.Priority`	Specifies the priority of the measurement. Valid priority levels are 0..15.

C++ structure:

```
typedef struct TSMW_IQIF_MEAS_CTRL
{           unsigned long NoOfSamples; unsigned char FilterType;
            unsigned char FilterID; unsigned char DataFormat;
            unsigned char AttStrategy; unsigned char Splitter;
            unsigned short Priority;
} TSMW_IQIF_MEAS_CTRL_t;
```

2.8.1.2 MeasCtrl values for streaming control

Description:

These fields configure the settings of the streaming file.

Field(s):

`StreamCtrl.StreamID`	Specifies the stream ID which shall be used to control this stream. Value range: 0..15
`StreamCtrl.StreamBufferSize`	Specifies the buffer size for streaming in MBs. At least 200 MB is recommended.
`StreamCtrl.MaxStreamSize`	Specifies the maximum streaming size in MB. **Note:** The stream data is automatically splitted into several files such that a maximum file size of 2GB is not exceeded.

C++ structure:

```
typedef struct TSMW_IQIF_STREAM_CTRL
{             unsigned long StreamID; unsigned long StreamBufferSize;
              unsigned long MaxStreamSize;
} TSMW_IQIF_STREAM_CTRL_t;
```

2.8.1.3 MeasCtrl values for capacity request for RF channels

Description:

These fields contains information about the requested receiver capacity and the desired observation time window.

Field(s):

`Frontends`	Specifies which frontend to use. 1: Frontend 1 2: Frontend 2 3: Both frontends
`Capacity`	Requested receiver capacity in % . The actual percentage a receiver resource will get is given by $\frac{\text{(Capacity of the receiver)}}{\text{(Sum of capacity of all resources)}}$ The sum capacity can be larger than 100%.
`ObservationTime`	Observation time window in ms. Only measurements that were taken in the last <*ObservationTime*> ms will count for the current resource usage

C++ structure:

```
typedef struct TSMW_IQIF_RESOURCE_PARAM
{             unsigned char Frontends; unsigned char Capacity;
              unsigned long ObservationTime;
}TSMW_IQIF_RESOURCE_PARAM_t;
```

2.8.1.4 MeasCtrl values for schedule control

Description:

The structure describes the schedule option for the scalable bandwidth management for the receivers. It has to be specified if the measurement request shall use a specific receiver resource.

Schedule structure do not exist

If no structure exist no receiver resource class will be used. Measurements that do not use a specific receiver resource class will not be considered for receiver capacity limitation!

Field(s):

`SchedCtrl.ResourceID`	Resource ID of the schedule task.
`SchedCtrl.SchedulerTimeConst`	Scheduler time constant for non-periodic measurements in ms. The priority of a non-periodic measurement is given by $\frac{<\text{Priority in meas.req.}> + <\text{Waiting time in ms}>}{<\text{SchedulerTimeConst}>}$ This means that the greater the constant is the longer is the waiting time of the standard measurement request.

C++ structure:

```
typedef struct TSMW_IQIF_SCHEDULE_CTRL
{           unsigned long ResourceID; unsigned long SchedulerTimeConst;
}TSMW_IQIF_SCHEDULE_CTRL_t;
```

2.8.1.5 MeasCtrl values for periodic measurement control

Description:

The structure describes the control structure for starting periodic measurement requests. It has to be specified if the measurement request has to be a periodic measurement request. Otherwise it is a standard measurement.

Field(s):

`Cmd`	Command: 0: When used within m-function TSMWIQMeasure: Start periodical measurement request. Otherwise: Add credits to periodical measurement request without changing parameters. 1: Clear all credits of periodical measurement request. De-activate the measurement request. 2: Cancel periodical measurement request. 3: Change parameters. 4: Adjust time slip. 5: Change attenuator and preamble settings.
`MeasRequestID`	Only used if Cmd == 3 (change parameters of the measurement request. Specifies the measurement request ID of periodical measurement request.
`IQPeriod`	Time period in I/Q samples. The minimum time period is 1ms -> 22000 I/Q samples.
`IQMinDistance`	Only used if no start time is given. Minimum distance between consecutive measurements in I/Q samples.
`GridTimeus`	Only used if start time is given. Time grid in microseconds on which measurements shall be performed. In that case, measurements are only performed on a time grid given by StartTime + n * GridTime. **Note:** Internally GridTimeus is converted from microseconds to I/Q-samples. The StartTime is always given in I/Q-samples.
`IQTimeSlip`	Only used if Cmd == 4 (Adjust time slip). Defines the grid at which a measurement will be started is adjusted by the IQTimeSlip. Time slip is only available for periodic measurement requests with a start time. **Note:** The time slip is given in I/Q samples before resampling.

`NoOfMeasurements`	Only used if Cmd == 0 Specifies the initial number of measurements to deliver until new "measurement credits".
`TransmitWindow`	Specifies the transmit window. A transmit window defines the number of measurements that are performed until the first measurements are transmitted to the host. This prohibits the R&S TSMW of making more measurements than can be transmitted to the PC cause by i. e. limited ethernet capacity. Value range 2..16.
`Att[2]`	Only used if Cmd == 5 (Change attenuator and preample settings) Contain the new attenuator setting for each channel.
`Preamp[2]`	Only used if Cmd == 5 (Change attenuator and preamble settings) Contain the new preample setting for each channel.

C++ structure:

```
typedef struct TSMW_IQIF_PERIOD_CTRL
{       unsigned long Cmd; unsigned long MeasRequestID;
        unsigned __int64 IQPeriod; unsigned __int64 IQMinDistance;
        unsigned __int64 GridTimeus; short IQTimeSlip;
        unsigned short NoOfMeasurements; unsigned char TransmitWindow;
        unsigned char Att[2]; unsigned char Preamp[2];
}TSMW_IQIF_PERIOD_CTRL_t;
```

2.8.1.6 MeasCtrl values for trigger control

Description:

The structure describes the control structure for trigger configuration.

Field(s):

`TriggerCtrl.Cmd`	Command: 0: When used within m-function TSMWIQMeasure: Start triggering for a measurement. 1: Stop triggered measurement. 2: Change parameters.
`TriggerCtrl.Mode`	Specifies the trigger mode. At the moment only 0 is available.
`TriggerCtrl.Falling`	Specifies the trigger edge: 0: Rising 1: Falling
`TriggerCtrl.TriggerLine`	1: Use trigger input 1 2: Use trigger input 2 3: Use trigger on both inputs.

`TriggerCtrl.MeasRequestID`	Only used if Cmd == 2 (change parameters) Specifies the measurement request ID of periodical measurement request
`TriggerCtrl.Att[2]`	Only used if Cmd == 2 (change parameters) Change the current attenuator setting for each channel.
`TriggerCtrl.Preamp[2]`	Only used if Cmd == 2 (change parameters) Change preample setting for each channel.

C++ structure:

```
typedef struct TSMW_IQIF_TRIG_CTRL
{         unsigned long Cmd; unsigned char Mode; unsigned char Falling;
          unsigned char TriggerLine; unsigned long MeasRequestID;
          unsigned char Att[2]; unsigned char Preamp[2];
}TSMW_IQIF_TRIG_CTRL_t;
```

2.8.1.7 MeasCtrl values for RF channel 1 and RF channel 2

Description:

Each RF channel has its own fields. These fields contain measurement control values for the corresponding RF channel.

Field(s):

The following explanation shows only the values for the RF channel 1 control fields.

They correspond to the RF channel 2. Therefore replace the prefix "ChannelCtrl1"(RF channel 1) to "ChannelCtrl2" (RF channel 2).

`ChannelCtrl1.Frequency`	Center frequency (Hz)
`ChannelCtrl1.UseOtherFrontend`	Reserved for future use, has to be set to zero
`ChannelCtrl1.NoOfChannels`	Specifies the number of channels that shall be used (1..4). The received bandwidth of the selected frontend is split up into the corresponding number of channels. I. e.: If 3 channels are used, the downsampling factor has to be at least 3.
`ChannelCtrl1.Attenuation`	Specifies the attenuation to use if the parameter is set. In auto attenuation mode this is the initial (suggested) attenuation to use for the measurement. **Note:** Only the values 0,5,10 and 15 dB are calibrated.

`ChannelCtrl1.Preamp`	Defines whether the preamp. shall be enabled or not. In auto attenuation mode, this is the initial (suggested) setting of the preamp. 0: Disable 1: Enable
ChannelCtrl1.CalibInput	Defines whether the calibration input shall fit to the used frontend. 0: Disable 1: Enable **Note:** The calibration amplifier has to be enabled, too.
`ChannelCtrl1.FreqShift[]`	Defines the frequency shift of each frequency shift from center frequency for each channel
`ChannelCtrl1.ChannelDelay[]`	Specifies a separate delay in I/Q samples after filtering and downsampling for each channel.
BlockSize	Reserved for future use, has to be zero.
BlockSkip	Reserved for future use, has to be zero.

C++ structure:

```
typedef struct TSMW_IQIF_CH_CTRL
{           unsigned __int64 Frequency; unsigned char UseOtherFrontend;
            unsigned char NoOfChannels; char Attenuation;
            unsigned char Preamp; unsigned char CalibInput;
            double FreqShift[4]; long ChannelDelay[4];
            long BlockSize; long BlockSkip;
} TSMW_IQIF_CH_CTRL_t;
```

2.8.2 Structure: FilterSpec

Description:

The structure describes the filter parameter structure.

The down sampling factor has to be greater than the no. of sub channels. The maximum down sampling factor is 100.

Therefore following requirement has to be fulfilled:

$$(\text{No. of Sub Channels}) < (\text{Downsampling Factor}) < \left(\frac{\text{Sampling Rate}}{\text{No. of Sub Channels}}\right)$$

Field(s):

FilterID	Specifies the filter number.
NoOfCoeffs	Specifies the number of filter coefficients.
OversplFact	Specifies the filter coefficient oversampling factor.
FilterCorrdB	Specifies the negative average filter gain in the pass band in dB. The result will be corrected by this value. **Note:** This value is the remaining filter gain after considering a factor of 2^(-ResultShiftNumber).
ResultShiftNumber	Specifies the number of bits the result is shifted. Compensates for the filter gain.
GroupDelay	Specifies the group delay of the current filter in taps. A measurement will automatically started "GroupDelay" samples before the given start time (if a start time was given). Default value is 0.
NDown	Specifies the fractional down sampling factor. **Note:** Since the maximum number of filter coefficients and the oversampling factor depends on the downsampling factor, it is essential that these values match, for correct filtering/downsampling. **Note:** The down sampling factor has to be greater than the no. of sub channels. The maximum down sampling factor is 100.

C++ structure:

```
typedef struct TSMW_IQIF_FILTER_PARAM
{           unsigned short FilterID; unsigned short NoOfCoeffs;
            unsigned char OvsplFact; double FilterCorrdB;
            unsigned char ResultShiftNumber; unsigned short GroupDelay;
            double Ndown;
} TSMW_IQIF_FILTER_PARAM_t;
```

2.8.3 Structure: TSMWOptions

Description:

The structure describes the Digital I/Q interface options for the R&S TSMW.

Field(s):

`Frontends`	Specifies which frontend to enable 1: Enable frontend 1 2: Enable frontend 2 3: Enable both frontends
`AMPS_CH1`	Enable the frequency band for RF channel 1. Set to (2^32-1) for all bands (amplifiers).
`AMPS_CH2`	Enable the frequency band for RF channel 2. Set to (2^32-1) for all bands (amplifiers).
`Mode`	At the moment only the value zero is supported.

C++ structure:

```
typedef struct TSMW_IQIF_MODE
{             unsigned char Frontends; unsigned long AMPS_CH1;
              unsigned long AMPS_CH2; unsigned char Mode;
}TSMW_IQIF_MODE_t;
```

2.8.4 Structure: TSMWIQResult

Description:

The structure describes the IQ result parameter structure.

Field(s):

`MeasRequestID`	Specifies the measurement request ID.
`Reserved`	Unused
`StartTimeIQ`	Specifies the I/Q-Counter value at which measurement was started. Note: Each individual channel will have a different start time, according to the specified ChannelDelay parameter in the measurement request structure.
`StartTimeHost`	Specifies the corresponding host time.
`Offset`	Offset in I/Q-Samples from measurement start (only for streaming meas.)
`NoOfSamples`	Number of samples that were copied for each sub channel
`Reserved2`	Unused

`Fsample`	Specifies the sampling rate (sample/s) that was actually used. Mostly the used sampling rate for a specific filter differs from the requested sample rate. This is caused by rounding errors. I. e.: For a specific filter the requested sample rate is 10Ms/s. But in this case the actually used sample rate for the filter is `10.00000772597512 MS/s`. This information is available via field Fsample.
`Attenuation.1`	Specifies the used attenuation in dB for frontend 1.
`Attenuation.2`	Specifies the used attenuation in dB for frontend 2.
`Preamp.1`	Specifies for frontend 1: 0: Preamp was off 1: Preamp was on
`Preamp.2`	Specifies for frontend 2: 0: Preamp was off 1: Preamp was on

C++ structure:

```
typedef struct TSMW_IQ_RESULT
{    unsigned long MeasRequestID; unsigned long Reserved;
     unsigned long long StartTimeIQ; unsigned long long StartTimeHost;
     unsigned long long Offset; unsigned long NoOfSamples;
     unsigned long Reserved2; unsigned long Attenuation[2];
     unsigned longPreamp[2];
} TSMW_IQ_RESULT_t;
```

2.8.5 Structure: StreamStatus

Description:

The structure describes the current status of the streaming measurement.

Field(s):

`Status`	Streaming status 0: Not activated 1: Running 2: Finished 3: Stopped (because of an error) 4: Stopped because maximum file size reached
`BlockSize`	Number of samples per streaming block.
`NoOfBlocks`	Number of received blocks.
`NoOfSkippedBlocks`	Number of skipped blocks.

C++ structure:

```
typedef struct TSMW_IQIF_STREAM_STATUS
{            unsigned char Status; unsigned long BlockSize;
             unsigned long NoOfBlocks; unsigned long NoOfSkippedBlocks;
} TSMW_IQIF_STREAM_STATUS_t;
```

2.8.6 Structure: StreamInfo

Description:

The structure shows information about date and time about a streaming measurement file.

Field(s):

`Year`	The year when the stream was started.
`Month`	The month when the stream was started.
`Date`	The date of the day when the stream was started.
`Hour`	The hour when the stream was started.
`Minute`	The minute when the stream was started.
`Second`	The second when the stream was started.
`Description[256]`	In this field the user can write a short description about the streaming file. This information is saved with the streaming file.
`NoOfFiles`	Number of files for this streaming measurement. Every time 2GB is reached a new files with increased index number will be created (i. e. myfile_000, myfile_001, etc.)
`StreamSize`	Total stream size in MB.
`BlockSize`	Number of samples per streaming block.
`NoOfBlocks`	Number of received blocks.
`NoOfSkippedBlocks`	Number of skipped blocks.

C++ structure:

```
typedef struct TSMW_IQIF_STREAM_INFO
{         unsigned long Month; unsigned long Date; unsigned long Hour;
          unsigned long Minute; unsigned long Second;
          char Description[256]; unsigned __int64 NoOfFiles;
          unsigned long StreamSize; unsigned long BlockSize;
          unsigned long NoOfBlocks; unsigned long NoOfSkippedBlocks;
} TSMW_IQIF_STREAM_INFO_t;
```

2.9 R&S TSMW I/Q Interface Demo Application

The "TSMW MATLAB I/Q Interface" script is an example how using the TSMW-K1 API. On the basis of the example implementation the functionality of the TSMW-K1 API will be explained.

Likewise, the demo application may be utilized as a skeletal structure for customized applications in order to quickly obtain the measurement results needed.

The demo application makes it possible to define the center frequency, the sampling rate and the measurement filter. In this context, the measurement filter is used primarily to define the bandwidth to be measured. Individual measurements or continuous measurements may be carried out. The user can also define which of the two R&S TSMW receivers is to be used. For professional applications, it is furthermore possible to set the attenuation values, the activation of the preamplifier, and the data format of the digital I/Q data.

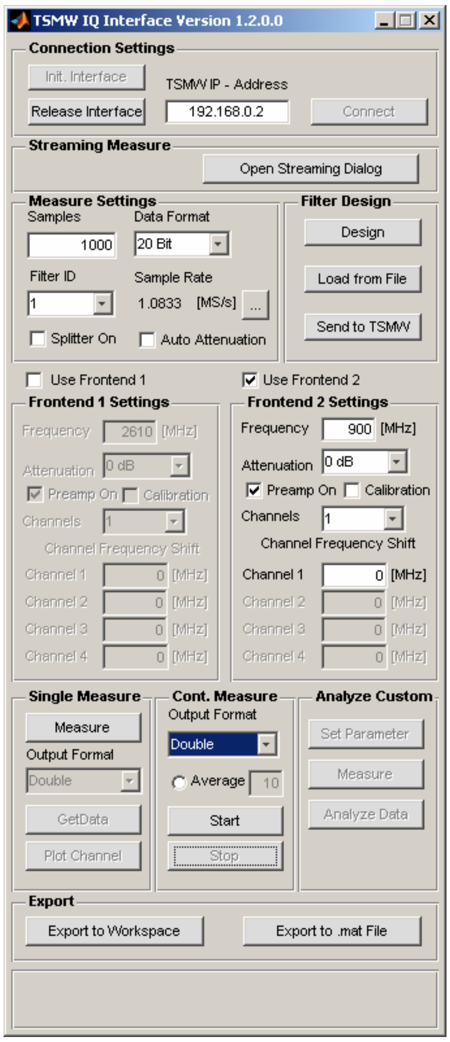
Source code of the R&S TSMW MATLAB IQ Interface application

The source code of the `TSMWIQInterfaceDemo.m` is available on the installed folder.

2.9.1 Main Window "TSMW Matlab IQ Interface"

The main window of the R&S TSMW MATLAB IQ Interface script consists of different parts.

On the left you can control the connection to the R&S TSMW and configure the measurement algorithm. Every button is mapped to one or more R&S TSMW-K1 MATLAB functions and shows an example implementation.



TSMW IQ Interface Version 1.2.0.0

Connection Settings

Init. Interface | TSMW IP - Address: 192.168.0.2 | Release Interface | Connect

Streaming Measure

Open Streaming Dialog

Measure Settings

Samples: 1000
Data Format: 20 Bit
Filter ID: 1
Sample Rate: 1.0833 [MS/s] ...
☐ Splitter On ☐ Auto Attenuation

Filter Design

Design | Load from File | Send to TSMW

☐ Use Frontend 1 ☑ Use Frontend 2

Frontend 1 Settings

Frequency 2610 [MHz]
Attenuation 0 dB
☑ Preamp On ☐ Calibration
Channels 1
Channel Frequency Shift
Channel 1 0 [MHz]
Channel 2 0 [MHz]
Channel 3 0 [MHz]
Channel 4 0 [MHz]

Frontend 2 Settings

Frequency 900 [MHz]
Attenuation 0 dB
☑ Preamp On ☐ Calibration
Channels 1
Channel Frequency Shift
Channel 1 0 [MHz]
Channel 2 0 [MHz]
Channel 3 0 [MHz]
Channel 4 0 [MHz]

Single Measure

Measure
Output Format: Double
GetData | Plot Channel

Cont. Measure

Output Format: Double
○ Average 10
Start | Stop

Analyze Custom

Set Parameter | Measure | Analyze Data

Export

Export to Workspace | Export to .mat File

Figure 2-7: Main window of TSMW IQ Interface demo application.

For viewing the measurement data a new window appears when a single measurement was done or a stream measurement is performing, see Figure 2-8: Example plot from "Frontend 2".

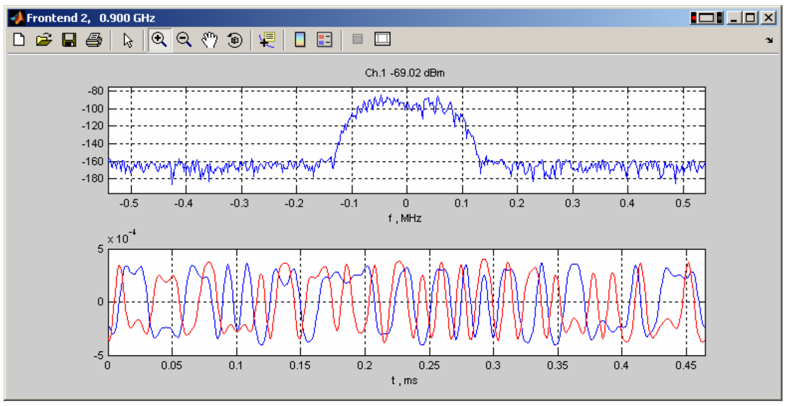


Figure 2-8: Example plot from "Frontend 2".

The following tables list the functionality and the corresponding MATLAB functions of the R&S TSMW MATLAB IQ Interface application.

Connection Settings

Name	Description
Init.Interface	Load the R&S TSMW IQ interface library `TSMWIQInterface.dll`. M-function: `TSMWInitInterface`
ReleaseInterface	Unload the external R&S TSMW IQ interface library and disconnect from TSMW(s). M-function: `TSMWReleaseInterface`
TSMW IP-Address	Enter the R&S TSMW IP address. Default value is `192.168.0.2`. M-parameter: `IPAddress`
Connect	Establish a connection to the R&S TSMW with the given IP address. M-function: `TSMWConnect`

Streaming Measure

Name	Description
Open Streaming Dialog	Open a dialog window which controls the streaming settings for I/Q data streaming measurements. For details see Streaming Dialog on page 41.

Measure Settings

Correspond to MeasCtrl.MeasCtrl. For details see Structure: MeasCtrl on page 24.

Name	Description
Samples	Specify the number of I/Q samples to measure. Field: `MeasCtrlMeasCtrl.NoOfSamples`
Data Format	Specify I/Q data compression format for block wise compression. Field: `MeasCtrl.MeasCtrl.DataFormat` **Note:** **Reduce the data transfer rate** To reduce the amount of data accumulated, the R&S TSMW-K1 interface offers to set the bit resolutions individual to 20 Bit, 16 Bit, 12 Bit or 8 Bit. Thus the user can reduce the transmission rate on the LAN interface.
Filter ID	Select a filter to use. 0 : Select to use a predefined filter. ≥ 1: Select to use a user defined filter. **Note:** A user defined filter has to be sent to the R&S TSMW before it is available in the drop down list. Fields: `MeasCtrl.MeasCtrl.FilterID`, `MeasCtrl.MeasCtrl.FilterType`
...	Display the transfer function of the selected filter.

Name	Description
Splitter On	Specify the RF channel 1 splitter. Splitting the signal after preseletor to both frontends. Activated: Enable splitterStructure MeasCtrlTemplate Not activated: Disable splitter Field: `MeasCtrl.MeasCtrl.Splitter`
Auto Attenuation (only for block measurements)	Specify the attenuation strategy. Activated: Auto attenuation. If activated and the signal strength is to strong: 1. The preamplifier is deactivated. 2. If the signal strength is still to strong, the attenuation will be increased in 5dB steps till the signal strength is ok. **Note:** After each reconfiguration step the I/Q measurement is restarted. Field: `MeasCtrl.MeasCtrl.AttStrategy`

Filter Design

Name	Description
Design	Open a dialog window to creates and modify filters for I/Q data measurements. For details see Filter Design Dialog on page 43.
Load from file	Loads a saved filter specification.
Send to TSMW	Transmit the last design filter to the R&S TSMW. Once a filter specification is transferred, it is available on the drop down menu "FilterID". M-function: `TSMWIQSetup`

Frontend 1 Settings/ Frontend 2 Settings

Correspond to MeasCtrl.ChannelCtrl1 / MeasCtrl.ChannelCtrl2.

Name	Description
Use Frontend 1	If checked, a measurement will be performed on frontend 1. This means that only the field ChannelCtrl1 is present in the MeasCtrl structure
Use Frontend 2	If checked, a measurement will be performed on frontend 2. This means that only the field ChannelCtrl2 is present in the MeasCtrl structure.

Name	Description
Frequency (in MHz)	Defines the center frequency. Field: `MeasCtrl.ChannelCtrl1.Frequency,` `MeasCtrl.ChannelCtrl2.Frequency`
Attenuation	Defines the attenuation to use. In the auto attenuation mode this is the initial (suggested) attenuation to use for the measurement. Field: `MeasCtrl.ChannelCtrl1.Attenuation,` `MeasCtrl.ChannelCtrl2.Attenuation,`

Name	**Description**
Preamp. On	Defines whether the preamp. shall be enabled or not. In auto attenuation mode, this is the initial (suggested) setting of the preamp. Field: `MeasCtrl.ChannelCtrl1.Preamp,` `MeasCtrl.ChannelCtrl2.Preamp`
Calibration	Defines whether the calibration input shall fit to the used frontend. Field: `MeasCtrl.ChannelCtrl1.CalibInput,` `MeasCtrl.ChannelCtrl2.CalibInput.`
Channels	Defines the number of channels that shall be used (1..4). The received bandwidth of the selected frontend is split up into the corresponding number of channels. Field: `MeasCtrl.ChannelCtrl1.NoOfChannels,` `MeasCtrl.ChannelCtrl2.NoOfChannels`
Channels Frequency Shift 1 to 4 (in MHz)	Defines a separate delay in I/Q samples after filtering and downsampling for each channel. Field: `MeasCtrl.ChannelCtrl1/2.FreqShift1 to 4,` `MeasCtrl.ChannelCtrl1/2.FreqShift1 to 4`

Single Measure
Performs a single I/Q data measurement.

Name	**Description**
Measure	Start a new measurement with above defined settings. M-function: `TSMWIQMeasure`
Output Format	Define the unpack format for the measurement result. M-functions: `TSMWIQGetDataInt16, TSMWIQGetDataInt32,` `TSMWIQGetDataSingle, TSMWIQGetDataDouble`
GetData	Transfer the measured data from R&S TSMW to the local computer.
Plot Channel	Show results.

Cont. Measure
Repeats the number of single measurement without deleting the I/Q data of the last measurement. This is particularly of interest when using the configurations "Average" or "Max Hold" to take previously recorded measurements into account for averaging / maximum search.

Name	**Description**
Output Format	Define the unpack format for the measurement result. M-functions: `TSMWIQGetDataInt16, TSMWIQGetDataInt32,` `TSMWIQGetDataSingle, TSMWIQGetDataDouble`
Average	If checked an average spectrum is calculated by averaging over the specified number of measurements.
Start	Start a continuous measurement immediately. M-function: `TSMWIQMeasure`
Stop	Stop a continuous measurement.

Analyze Custom

For all elements a corresponding dummy function in file `TSMWIQInterfaceDemo.m` exists.

Name	Description
Set Parameter	M-function in `file TSMWIQInterfaceDemo.m` can be customized.
Measure	M-function in `file TSMWIQInterfaceDemo.m` can be customized
Analyze Data	M-function in file `TSMWIQInterfaceDemo.m` can be customized.

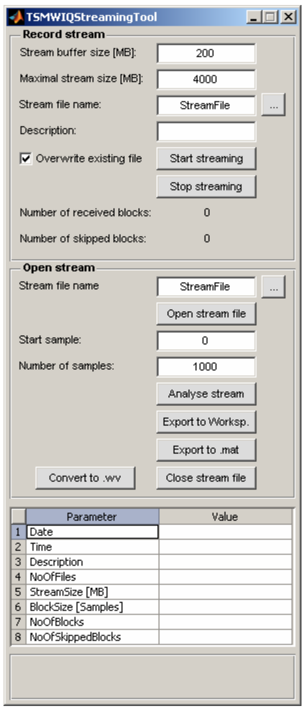
Export

Name	Description
Export to Worksp.	Export the measured data to the MATLAB Workspace.
Export to .mat	Export the settings to a *.mat File.

2.9.2 Streaming Dialog

The “TSMW I/Q Streaming Tool" dialog window controls the streaming settings for I/Q data streaming measurements.

The TSMWIQStreamingTool dialog window set and shows the parameters for streaming measurements. Furthermore a streaming measurement is controlled and analyzed over this window.


TSMWIQStreamingTool

Record stream

Stream buffer size [MB]: 200

Maximal stream size [MB]: 4000

Stream file name: StreamFile

Description:

☑ Overwrite existing file

Number of received blocks: 0

Number of skipped blocks: 0

Open stream

Stream file name StreamFile

Start sample: 0

Number of samples: 1000

	Parameter	Value
1	Date	
2	Time	
3	Description	
4	NoOfFiles	
5	StreamSize [MB]	
6	BlockSize [Samples]	
7	NoOfBlocks	
8	NoOfSkippedBlocks	

Figure 2-9: "TSMW IQ Streaming Tool" dialog window

The following table describes the functionality and the corresponding MATLAB functions of the dialog window.

Record stream

Name	Description
Stream buffer size [MB]	Specifies the stream buffer. Choose a value at least 200MB. Default value is `200MB`. Field: `MeasCtrl.StreamCtrl.StreamBufferSize`
Maximal stream size [MB]	Specifies the maximum streaming size in MB. Field: `MeasCtrl.StreamCtrl.MaxStreamSize` **Note:** The stream data is automatically splitted into several files such that a maximum file size of 2GB is not exceeded.
Stream file name	Specifies the file where the streaming data has to be saved. M-function: `TSMWIQStream,` parameter: `FileName`
Description	In this field the user can write a short description about the streaming file. This information is saved with the streaming file. Field: `StreamInfo.Description`
Overwrite existing file	Specifies whether an already existing streaming data file will be overwritten. M-function: `TSMWIQStream`, parameter: `CreateIfExists`
Start streaming	Start a streaming measurement and save the I/Q data in the specified file. M-function: `TSMWIQStream`
Stop streaming	Stop the current streaming measurement. M-function: `TSMWIQStopStreaming`

Open stream

Name	Description
Stream file name	Specifies the streaming file to view. M-function: `TSMWIQOpenStreamFile`, parameter: `FileName`
Open stream file	Open the streaming file. Furthermore information about parameter values is shown inside the dialog window. M-function: `TSMWIQOpenStreamFile`
Start sample	Index of I/Q sample where to start.
Number of samples	Enter the number of samples for analysis
Analyze stream	Starts the analysis of the streaming data.(Plot spectrum and I/Q components of selected segment of the recorded I/Q stream.)
Export to .Worksp.	Export the stream data to the MATLAB Workspace.
Export to .mat File	Export the stream date to *.mat file format.
Convert to .wv	Convert streamed data to R&S Signal Generator waveform and list format in order to allow RF-replay on R&S Signal Generators. Therefore the correspon-ding signal is normalized prior to conversion in order to achieve best replay performance. The necessary sig. generator level for regeneration at exact the same signal level is written into the comment section of the resulting `*.wv` file.
Close stream file	Close the open stream file. M-function: `TSMWIQCloseStreamFile`

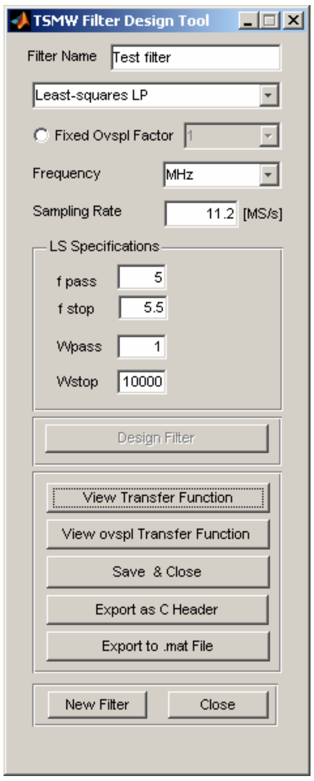
2.9.3 Filter Design Dialog

The "TSMW Filter Design Tool" dialog window creates and modifies filters for I/Q data measurements. The designed filters can be saved as .mat file for loading it back into the Demo application. Furthermore the designed filters can be saved as C++ header file and could be included it into a C++ user application.

The drop down menu offers several filter design algorithms with predefined values.

The filter designer base on the MATLAB Filter Design functions.

It is possible to check the new designed filter over the "View transfers function" functionality. The black vertical line in the "Filter Response" window corresponds to `Sampling Rate / 2`, i.e. the maximum frequency without aliasing.



TSMW Filter Design Tool

Filter Name: Test filter

Least-squares LP

Fixed Ovspl Factor: 1

Frequency: MHz

Sampling Rate: 11.2 [MS/s]

LS Specifications

f pass: 5

f stop: 5.5

Wpass: 1

Wstop: 10000

Figure 2-10: "TSMW Filter Design Tool" dialog window

The following table describes the functionality and the corresponding MATLAB functions of the "TSMW Filter Design Tool" dialog window.

Name	**Description**
Filter Name	Defines a filter name.
-/-	Select the desired filter type.
Fixed Ovspl Factor	If checked a filter coefficient oversampling factor can be specified Field: `FilterSpec.OvsplFact`

Name	**Description**
Design Filter	A new filter with the entered values will be created.
View Transfer Function	The function checks the designed filter.
View ovrspl Transfer	The function checks the designed filter.
Save & Close	Save the designed filter and return to the main window of the MATLAB Demo application. The designed filter is available for measure purpose on the current instance of the MATLAB Demo application.
Export as C Header	Export the designed filter to a C Header file.
Export to .mat File	Export the designed filter to an M-file.
New Filter	Reset the values and start to design a new filter.
Close	Close the "TSMW Filter Design Tool" window without saving and return to the main window.

2.9.4 Use R&S TSMW MATLAB IQ Interface functions for user specific MATLAB scripts

The R&S TSMW MATLAB IQ Interface demo script contains a couple of functions which could be adapted for user specific MATLAB scripts.

2.10 How to...

This chapter describes typical use cases for the R&S TSMW MATLAB IQ Interface demo application.

2.10.1 Use the R&S TSMW MATLAB IQ Interface Application (compiled)

Prerequisite:

- The TSMW-K1 software is installed on the PC.
  For detailed information how to install refer to chapter Install on page 11.
- The MCR environment is installed.
- The R&S TSMW is successful connected to the PC.

Procedure:

► Select "Start"->"Programs"->"Rohde&Schwarz"->"TSMW-K1 <ver. no.>" -> "TSMW MatlabDemo IQ".

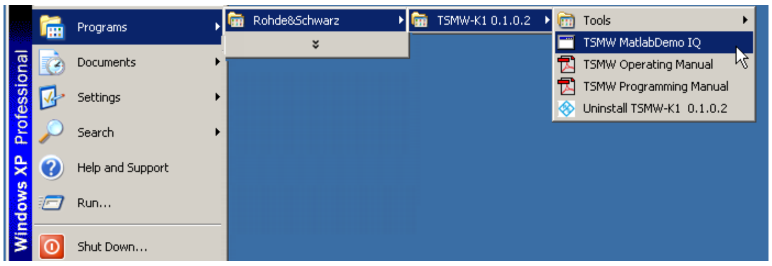


Figure 2-11: Start demo application

The main window of the R&S TSMW MATLAB IQ Interface application opens.

2.10.2 Use the R&S TSMW Matlab IQ Interface Application (not compiled)

Procedure:

1. Start MATLAB.
2. On the command window type : *“TSMWIQInterfaceDemo”*

The R&S TSMW Matlab IQ Interface application starts and the main window of the application open.

2.10.3 Use native C++ Interface

Procedure:

► Copy all files from

```
C:\Program Files\MATLAB\R2007a\bin\toolbox\RS_TSMWIQInterface\lib
```

including subfolders to your C++ user application folder.

2.10.4 Connect to the R&S TSMW

Prerequisite:

- A LAN connection between the computer and the R&S TSMW is required. How to link the software to R&S TSMW refer to the corresponding R&S TSMW operation manual.

Procedure:

1. Start the R&S TSMW MATLAB IQ Interface application.
   The R&S TSMW MATLAB IQ Interface script window opens.
2. Click on the "Init. Interface" button.
   An entry field for the R&S TSMW IP address appears.
3. Enter the IP address of the R&S TSMW device.
4. Click the "Connect" button.

A connection to the R&S TSMW is established.

2.10.5 Create a resampling filter

The following example demonstrates how to use the TSMW Filter Design Tool to design a user specific resampling filter in order to get a user specific sampling rates.

In this example, the desired signal is a WiMAX signal using a 1024-FFT and 10MHz bandwidth, specified in IEEE 802.16. The Sampling rate is specified with 11.2 MS/s, which means that after resampling to 11.2MS/s the baseband signal spans +/-5.6 MHz.

The WiMAX standard for a 1024-FFT signal, defines a guard interval of 92 sub carriers on the left and 91 on the right. That means the signal of interest is between `+/-(512- 91)*11.2/1024 MHz.= 4.6047MHz`

Hence a sensible choice for the pass band frequency is 4.6MHz. To sufficiently suppress interferences caused by adjacent channels, the stop band edge is chosen to be the edge of the baseband signal, i.e., 5.6 MHz.

Sampling Rate

The minimum supported sampling rate is 0.22MS/s. However, best filtering results will be achieved with sampling rates above 1MS/

Prerequisites:

- The R&S TSMW MATLAB IQ Interface application is running.
- A connection to a R&S TSMW is established.

Procedure:

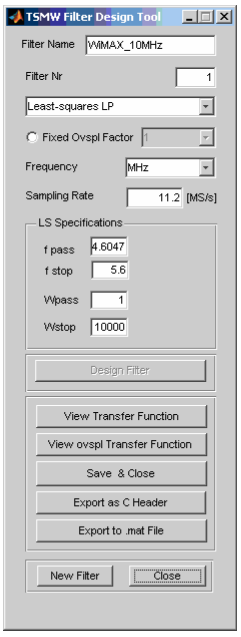
1. On the “TSMW MATLAB IQ Interface” window click the "Design" button.
   The Filter Design GUI opens.

2. From the “-select Filtertype-“ button choose “Least squares Lowpass”.
   Now a parameter input section appears, where the input parameters for the specific design can be defined.

   Note:
   A filter design using the “Modified Equiripple Lowpass” or “Raised Cosine Lowpass” function is also possible, but with the restrictions on low- , and passband edges, the “Least squares Lowpass” will usually give the best design.

3. Enter the parameters:
   For this example as follow (see figure):

   a. Set the Sampling rate to 11.2 [MS/s],
   b. Enter the passband frequency (f pass)
   c. Enter the stopband frequency (f stop).
   d. The parameters wpass and wstop are the weighting factors for the pass band and stop band.
      Decreasing the weighting factor for the stop band will decrease the stop band attenuation, increasing it will increase it and cost a higher pass band ripple. Sensible values for the stop band weighting factor are 1000 to 10000. The pass band weighting factor will usually be 1.



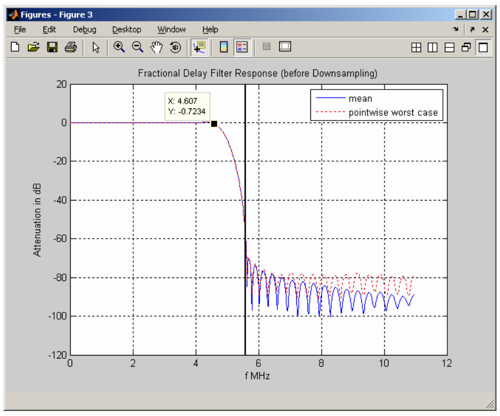
Filter Name: WiMAX_10MHz
Filter Nr: 1
Least-squares LP
Fixed Ovspl Factor: 1
Frequency: MHz
Sampling Rate: 11.2 [MS/s]
LS Specifications
f pass: 4.6047
f stop: 5.6
Wpass: 1
Wstop: 10000

Figure 2-12: Design filter for WiMAX measurements

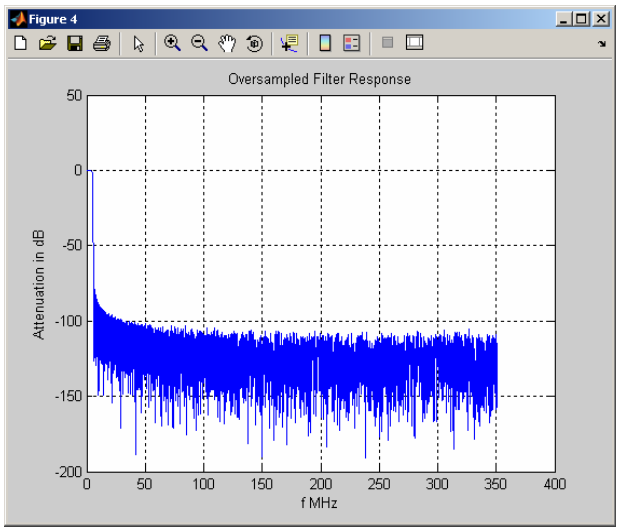
4. If the parameters are selected, as depicted in figure and step 3, press the “Design Filter” button to start the filter designer.

A new filter is created and new buttons appears with further actions on the designed filter.

View Transfer Function	To check the designed filter, click “View Transfer Function” The transfer function plot appears. It shows the filter response of the fractional filter before downsampling. The black line in the transfer function plot shows ½ of the sampling frequency, which is the maximum input frequency that does not produce aliasing products. In the figure below the marker is set to the passband edge.



View ovspl Transfer Function	To check the designed filter, click “View ovspl Transfer Function” With this plot you can verify if the designed filter fulfils the requirements. The plot shows the impulse response of the oversampled transfer function as depicted in the picture below.



Save & Close	Pass the designed filter to the current instance of R&S TSMW MATLAB IQ Interface application.
Export as C Header	Save and export the designed filter to a C Header file. Thus it is possible to load the designed filter into C++ applications, afterwards.

Export to .mat File	Save and export the designed filter to a *.mat file. Thus it is possible to load the designed filter into the R&S TSMW IQ Interface Demo Application, afterwards.

A new filter is created.

Submit filter

To make a filter selectable you have to submit the filter explicitly to R&S TSMW. How to submit the filter see Use predefined filters on page 49.

2.10.6 Use predefined filters

Prerequisites:

- The Demo GUI application is running.
- A connection to the R&S TSMW is established.

Procedure:

1. Click the “Load from File” to load a filter or design a new filter over the “Design” button. To design a new filter see procedure Create a resampling filter.
2. Click the "Send to TSMW" button.

The filter is transferred to the R&S TSMW and selectable over the “Filter ID” drop down menu for use.

2.10.7 Modify the measurement values

Prerequisites:

- The R&S TSMW MATLAB IQ Interface application is running and a connection to the R&S TSMW is established.

Procedure:

1. Edit the general values in the group box "Measure Settings".
   Please refer to chapter Structure: MeasCtrl on page 24 for an explanation of the parameters.
2. Check one check box “Use Frontend 1” or “Use Frontend 2” to activate the corresponding group box “Frontend 1 Settings” or “Frontend 2 Settings”.
3. Edit the values in the activated group box “Frontend 1 Settings” or “Frontend 2 Settings”.
   Please refer to chapter Structure: MeasCtrl on page 24 for an explanation of the parameters.
   a. Before a filter can selected in the “Filter ID.” drop down menu, it has to be designed.
      How to design a filter and make it available are explained in procedure Create a resampling filter and Use predefined filters on page 49.

The measurement settings are modified.

Now a measurement can be performed.

2.10.8 Perform a continuous measurement

Prerequisites:

- The R&S TSMW MATLAB IQ Interface application is running and a connection to the R&S TSMW is established.
- The measurement settings are defined.

Procedure:

1. It is possible to customize the average result of a number of measurements.
   a. Therefore click on the "Average Result" check box.
      The "No. of. Meas." entry field will be activated.
   b. Enter the desired number of measurements for the generation of average results.
2. Select from the drop down list box "Output Format" the desired output width.
3. Click the "Start" button to start the continuous measurement.
   The measurement starts and the data are available on the application.
4. Click the "Stop" button to stop the measurement.

A measurement was performed.

2.10.9 Perform a single measurement

Prerequisites:

- The R&S TSMW MATLAB IQ Interface application is running and a connection to the R&S TSMW is established.
- The measurement settings are defined.

Procedure:

1. Click the "Measure" button.
   The R&S TSMW starts a measurement. Until now no measurement data are transferred to the MATLAB application.
2. Select from the drop down list box "Output Format" the desired output width.
3. Click the "GetData" button to transfer the measurement result to the application.
4. Click the “Plot Channel” button to visualize the measurement.

A measurement was performed.

2.10.10 Record streaming measurement data

Prerequisites:

- The“ R&S TSMW MATLAB IQ Interface application is running and a connection to the R&S TSMW is established.
- The measurement settings in the main window are defined.
- The possible maximum bandwidth is well-known.

Therefore calculate the maximum I/Q data rate and compare it to the supported maximum bandwidth of the used computer equipment. For more details see Additional Requirements for I/Q Streaming on page 10.

Procedure:

1. Click the "Open Streaming Dialog" button.
   The TSMWIQStreamingTool dialog window opens.

   Note:

   The parameters for the streaming measurement will be taken from the main TSMW IQ Interface GUI at the time when the streaming dialog opens. This means that in order to start a new stream measurement, you have to close the streaming dialog, change the measurement parameters and reopen the streaming dialog again.
2. Inside the "Record stream" group fill-out the following fields:
   a. Enter the stream buffer size. It has to be at least 200 MB.
   b. Enter the maximum stream size. The maximum value is 4000 MB.
   c. Enter a stream file name
   d. Enter a short description.
3. Click "Start streaming" to start the measurement.
   The streaming measurement is started.
4. Click "Stop streaming" to stop recording.

A streaming measurement was performed and the measured data are saved in a file for further analysis.

2.10.11 View streaming measurement data

Prerequisites:

- The R&S TSMW MATLAB IQ Interface application is running.
- Recorded data of a streaming measurement exists.

Procedure:

1. Click the "Open Streaming Dialog" button.
   The TSMWIQStreamingTool dialog window opens.
2. Inside the "Open stream" group:
   a. Enter the name of the streaming file into the field "Stream file name".
   b. Click "Open stream file".
      Information about the streaming file appears at the end pf the window.
   c. Enter the number of start samples into the field "Number of start sample".
   d. Enter the number of samples into the field "Number of samples".
   e. Click "Analyze stream"
      A plot appears to analyze the data.
   f. Click "Close stream file".
      The streaming file is closed.

2.10.12 Export single data

Prerequisites:

- The R&S TSMW MATLAB IQ Interface application is running.
- A single measurement or a continuous measurement was performed so that a measured data block is loaded for analysis on the application.
- A MATLAB Workspace is open (only required for "Export to Workspace").

Procedures:

Export to MATLAB Workspace

► On the main window of the application click "Export to the Workspace".

The measured data block is exported to the current open MATLAB Workspace.

Export to .mat file

► On the main window of the application click "Export to the Workspace".

The measured data block is exported to the current open MATLAB Workspace.

2.10.13 Export streamed data

Prerequisites:

- The R&S TSMW MATLAB IQ Interface application is running.
- Recorded data of a streaming measurement exists.
- A MATLAB Workspace is open (only required for "Export to Workspace")

Procedure:

Export to MATLAB Workspace

1. Click the "Open Streaming Dialog" button.
   The TSMWIQStreamingTool dialog window opens.
2. Inside the "Open stream" group:
   a. Enter the name of the streaming file for export into the field "Stream file name".
   b. Click "Open stream file".
      Information about the streaming file appears at the end pf the window.
   c. Click "Export to Worksp.".
      The requested number of samples (parameters "Start sample" and "Number of samples") are exported into the current running MATLAB Workspace.
   d. Click "Close stream file".
      The streaming file is closed.

Export to .mat file

1. Click the "Open Streaming Dialog" button.
   The TSMWIQStreamingTool dialog window opens.
2. Inside the "Open stream" group:
   a. Enter the name of the streaming file for export into the field "Stream file name".
   b. Click "Open stream file".
      Information about the streaming file appears at the end pf the window.
   c. Click "Export to *mat".
      The requested number of samples (parameters "Start sample" and "Number of samples") are exported into a *. mat file.
   d. Click "Close stream file".
      The streaming file is closed.

2.10.14 Release the connection to the R&S TSMW IQ interface

Prerequisites:

- The R&S TSMW MATLAB IQ Interface script is running and a connection to the R&S TSMW is established.
- All measurements are stopped.

Procedure:

► Click the "ReleaseInterface" button.

The connection to the R&S TSMW is released.

2.10.15 Perform streaming measurements (C++ API)

Examples how to perform streaming measurements with the C++ API are available on the CD-ROM and inside the folder

`<R&S TSMW-K1 installation directory>\Examples\CPP.`

C++ example file	Description
`TestStreaming`	Performs online streaming. That mean during streaming the measured data are directly accessible for further processing.
`TestStreamingFile`	Performs streaming measurements into a file. After finishing the measurement, the file with the streamed data is accessible for further processing.

2.11 Troubleshooting

2.11.1 Working with a second R&S TSMW

Description

After replacing the R&S TSMW with another one the computer do not recognize the new connected R&S TSMW.

Possible reason:

The Address Resolution Protocol Cache (ARP-Cache) on the computer is not updated.

The ARP-Cache of the operating system has not updated the IP-to-Physical address translation tables with the physical address of the new connected R&S TSMW. The table still keeps the physical address of the R&S TSMW which was connected before.

Proposal for solution

On the computer open a command prompt and enter following command:

```
arp -d
```

The command deletes all entries in the IP-to-Physical address translation tables used by ARP.

Automatically update of the ARP-Cache
Usually the ARP-Cache will be refreshed by the operating system within 15 minutes.

2.11.2 Problems during LAN connection to the TSMW

Description

Over the Ethernet card no connection is possible between R&S TSMW and computer.

Possible reason

The computer firewall blocks the connection.

Proposal for solution

On the computer deactivate the firewall rules for the Ethernet card which is connected to the R&S TSMW.

If the problem still exists, check all installation settings of the R&S TSMW. Also see the R&S TSMW Operation Manual chapter 11 "Troubleshooting" for more details about troubleshooting the LAN connection.

2.11.3 Skipped I/Q data blocks during IQ streaming

Description

The field "Number of skipped blocks" shows an increasable high value of skipped I/Q data blocks.

Possible reason

The available hardware is to slow for the I/Q data streaming.

That means that during the transfer of the data from R&S TSMW to computer I/Q data blocks are skipped.

Proposal for solution

- Check the used hardware if it fit the system requirements. For details about the system requirements see chapter Additional Requirements for I/Q Streaming on page 6.
- Calculate the IQ data transfer rate needed and compare with your hardware capabilities
- Decrease the bit resolution used for starting the measurement
- Decrease the sampling rate
- Decrease the number of activated channels/frontends

Appendix

A C++ Header Files

A.1 TSMWIQInterfaceTypes.h

```
/**
  * TSMW Project   TSMW IQ Interface DLL
  *
  * @file         TSMWIQInterfaceTypes.h
  * @abstract
  *
  * @copyright    (c) 2007 Rohde & Schwarz GmbH & Co. KG, Munich
  * @author       Markus Herdin, Johannes Dommel
  * @version
  *    06.11.2007  Hd start
  *
  */

 #ifndef  _TSMWIQINTERFACETYPES_H
 #define  _TSMWIQINTERFACETYPES_H

 #define IQSAMPLINERATE_HZ    (395e6 / 18)

 // TSMW IQ interface mode structure
 typedef struct TSMW_IQIF_MODE
 {
   unsigned char    Frontends;            // Specifies which frontend to enable
                                   //   1: Enable frontend 1
                                   //   2: enable frontend 2,
                                   //   3: enable both frontends
   unsigned long    AMPS_CH1;             // A combination of bits that define which preselector bands (amplifier)
                                   // to activate for rf channel 1. Each bit corresponds to a specific frequency band
                                   //   Bit 0: <0.6GHz
                                   //   Bit 1: 0.6 - 1.2GHz
                                   //   Bit 2: 1.2 - 1.7GHz
                                   //   Bit 3: 1.7 - 2.5GHz
                                   //   Bit 4: >2.5GHz
   unsigned long    AMPS_CH2;             // Same for rf channel 2
   unsigned char    Mode;                // 0: Standard mode, 1: Calibration mode (shall not be used)
 } TSMW_IQIF_MODE_t;


 // Filter parameter structure
 typedef struct TSMW_IQIF_FILTER_PARAM
 {
   unsigned short   FilterID;            // ID of the filter
   unsigned short   NoOfCoeffs;           // Number of filter coefficients
   unsigned char    OvsplFact;            // Filter coefficient oversampling factor
   double           FilterCorrdB;         // Filter gain correction value in the passband in dB. The result will be corrected by this value.
                                   // NOTE: This value is the (negative) remaining filter gain after considering a factor
                                   // of 2^(-ResultShiftNumber)
   unsigned char    ResultShiftNumber;      // No of bits the result is shifted. Compensates for the
 filter gain.
```

```
  unsigned short   GroupDelay;          // Group delay of this filter in taps
  double           Ndown;               // Downsampling factor this filter was designed for.
} TSMW_IQIF_FILTER_PARAM_t;

// Measurement control structure
typedef struct TSMW_IQIF_MEAS_CTRL
{
  unsigned long    NoOfSamples;         // Number of IQ samples to measure
  unsigned char    FilterType;          // Filter type:
                                        //  0: Use predefined filters
                                        //  1: Use userdefined filters (they have to be sent to the TSMW, beforehand)
  unsigned short   FilterID;            // ID of the filter that shall be used
  unsigned char    DataFormat;          // IQ-data compression format for blockwise compression
                                        //  0: 2 x  8 Bit per complex sample
                                        //   1: 2 x 12 Bit
                                        //  2: 2 x 16 Bit
                                        //  3: 2 x 20 Bit
  unsigned char    AttStrategy;         // Attenuation strategy, currently unused, shall be set to zero
  unsigned char    Splitter;            // RF channel 1 splitter to split signal after preselector to
                                        // both frontends.
                                        //  0: Disable splitter
                                        //  1: Enable splitter
  unsigned short   Priority;            // Relative priority of this meas.req., Valid range: 0 .. 15,
                                        // 0 is lowest priority, 15 highest
} TSMW_IQIF_MEAS_CTRL_t;

// Channel control structure
typedef struct TSMW_IQIF_CH_CTRL
{
  unsigned __int64  Frequency;          // Center frequency in Hz
  unsigned char    UseOtherFrontend;    // Reserved for future use, has to be zero
  unsigned char    NoOfChannels;        // Number of channels that shall be used (1..4). This means the
receive
                                        // bandwidth of the selected frontend is split up into the corresponding
                                        // number of channels.
                                        // NOTE: The downsampling factor has to be larger or equal to the
                                        // number of channels. I.e. if 3 channels are used, the downsampling
                                        //  factor has to be at least 3.
  char             Attenuation;         // Attenuation to use (0..15dB)
                                        // NOTE: Only the values 0, 5, 10 and 15dB are calibrated
  unsigned char    Preamp;              // Defines whether the preamp shall be enabled or not
                                        //  0: Disable
                                        //   1: Enable
  unsigned char    CalibInput;          // Defines whether the calibration input shall fed to the used
frontend
                                        //  0: Disable
                                        //  1: Enable
                                        // NOTE: The calibration amplifier has to be enabled, too (see TSMW_IQIF_MODE)

  double           FreqShift[4];        // Frequency shift from center frequency in Hz for each subchannel
  long             ChannelDelay[4];     // Individual delay in taps for each subchannel (after
filtering/resampling)

  long             BlockSize;           // Reserved for future use, has to be zero
  long             BlockSkip;           // Reserved for future use, has to be zero
} TSMW_IQIF_CH_CTRL_t;

// IQ result parameter structure
typedef struct TSMW_IQIF_RESULT
{
  unsigned long      MeasRequestID;       // Meas.request ID

  unsigned long      Reserved;
  unsigned __int64   StartTimeIQ;            // IQ-Counter value at which measurement was started.
                                        // NOTE: Each individual channel will have a different start time,
                                        // according to the specified "ChannelDelay" parameter in the
                                        // meas.request structure.

  unsigned __int64   StartTimeHost;       // Corresponding host time

  unsigned __int64   Offset;                 // Offset in IQ-samples from measurement start (only
for streaming meas.)
  unsigned long      NoOfSamples;       // Number of samples that were copied for each subchannel
  unsigned long      Reserved2;
  double             Fsample;           // Sampling rate that was actually used

  unsigned long      Attenuation[2];       // Attenuation that was used in dB
  unsigned long      Preamp[2];          // 0: Preamp was off, 1: Preamp was on
} TSMW_IQIF_RESULT_t;

// Streaming control structure
typedef struct TSMW_IQIF_STREAM_CTRL
{
  unsigned long       StreamID;          // Stream ID, valid range: 0..15
  unsigned long       StreamBufferSize;  // Buffer size for streaming in MBytes, 200MB is recommended
  unsigned long       MaxStreamSize;     // Maximum streaming size in MBytes. Stream output files are
} TSMW_IQIF_STREAM_CTRL_t;

typedef struct TSMW_IQIF_STREAM_STATUS
{
  unsigned char       Status;              // Streaming status
                                        // 0:    Not activated
                                        // 1:    Running
                                        // 2:    Finished
                                        // 3:    Stopped because of error
                                        // 4:    Maximum streaming (file) size reached

  unsigned long       BlockSize;           // No of samples per streaming block
  unsigned long       NoOfBlocks;        // Number of received blocks
  unsigned long       NoOfSkippedBlocks; // Number of skipped blocks
} TSMW_IQIF_STREAM_STATUS_t;

// Streaming information structure
typedef struct TSMW_IQIF_STREAM_INFO
{
  // Date and time when streaming was started
  unsigned long       Year;
  unsigned long       Month;
  unsigned long       Date;
  unsigned long       Hour;
  unsigned long       Minute;
  unsigned long       Second;

  char                Description[256];  // Description

  unsigned __int64    NoOfFiles;            // Number of files for this stream
  unsigned long       StreamSize;        // Total stream size in MB

  unsigned long          BlockSize;          // No of samples per streaming block
  unsigned long          NoOfBlocks;         // Number of received blocks
```

```
  unsigned long        NoOfSkippedBlocks; // Number of skipped blocks
} TSMW_IQIF_STREAM_INFO_t;

typedef struct TSMW_IQIF_RESOURCE_PARAM
{
  unsigned char     Frontends;             // Specifies which frontend to use
                         //   1: Frontend 1
                         //   2: Frontend 2,
                         //   3: Both frontends
  unsigned char   Capacity;              // Requested receiver capacity in %. The actual percentage
                         // a resource will get is given by
                         // (Capacity / <Sum capacity of all resources>)
                         // The sum capacity can be larger than 100%
  unsigned long     ObservationTime;       // Observation time window in ms. Only measurements that
were
                         // taken in the last <ObservationTime> ms will count for
                         // the current resource usage.
} TSMW_IQIF_RESOURCE_PARAM_t;

typedef struct TSMW_IQIF_SCHEDULE_CTRL
{
  unsigned long   ResourceID;       // Resource ID the scheduler shall use
  unsigned long    SchedulerTimeConst;    // Scheduler time constant for non-periodic
measurements in ms
                         // The priority of a non-periodic measurement is given by
                         // <Priority in meas.req.> + <Waiting time in ms> / <SchedulerTimeConst>.
                         // This means that the priority of a meas.req. is increased with
                         // longer waiting time.
} TSMW_IQIF_SCHEDULE_CTRL_t;

typedef struct TSMW_IQIF_PERIOD_CTRL
{
  unsigned long   Cmd;             // Command:
                         //  0: Start period. meas.req. (when used in TSMWIQMeasureExt_c)
                         //      Otherwise, add credits to period. meas.req. without changing
parameters
                         //  1: Clear all credits of period. meas.req (de-activate it)
                         //  2: Cancel period. meas.req.
                         //  3: Change parameters
                         //  4: Adjust time slip
                         //  5: Change attenuator and preamp setting
  unsigned long    MeasRequestID;         // Meas.request ID of period. meas.req., only used
when changing
                         // parameters of a period. meas.req.
  unsigned __int64 IQPeriod;             // Time period in IQ samples. Minimum time period is 1ms -
> 22000 I/Q samples
  unsigned __int64 IQMinDistance;         // Minimum distance between consecutive measurements
in I/Q samples,
                         // only used if no start time is given.
  unsigned __int64 GridTimeus;         // Time grid in microseconds on which measurements shall be performed. Is only
used
                         // when a start time is given. In that case, measurements are only performed on a time
                         // grid given by StartTime + n * GridTime.
                         // NOTE: GridTimeus is converted, internally, from microseconds to I/Q-
samples.
                         // The StartTime is always given in I/Q-samples
  short            IQTimeSlip;            // Time slip for periodic measurement requests with a
start time, only
                         // used when Cmd == 4. The grid at which a measurement will be started is
                         // adjusted by the IQTimeSlip. NOTE: The time slip is given in I/Q samples
```

```
                                // before resampling.
  unsigned short   NoOfMeasurements;        // Number of measurements to deliver, initially, until new
                                // "measurement credits" have to be added via Cmd = 0.
  unsigned char    TransmitWindow;       // Transmit window (between 2 and 16). Gives the number of measurements
                                // that are performed until the first has to be transmitted to the host.
                                // This prohibits the TSMW of making more measurements than can be
                                // transmitted to the host (because of limited ethernet capacity).
  unsigned char    Att[2];              // New attenuator setting for each channel (only used when Cmd == 5)
  unsigned char    Preamp[2];             // New preamp setting for each channel (only used when
Cmd == 5)
} TSMW_IQIF_PERIOD_CTRL_t;

typedef struct TSMW_IQIF_TRIG_CTRL {
  unsigned long    Cmd;                     // 0: Start triggering (when used in TSMWIQMeasureTrig
command)
                                // 1: Stop triggered measurement
                                // 2: Change attenuator and preamp setting
  unsigned char    Mode;                  // Trigger mode, has to be zero
  unsigned char    Falling;                 // Trigger edge, 0: rising, 1: falling
  unsigned char    TriggerLine;              // 1: Use trigger input 1
                                // 2: Use trigger input 2
                                // 3: Trigger on both inputs
  unsigned long    MeasRequestID;            // Meas.request ID of period. meas.req., only used when
changing
                                // parameters of a triggered measurement
  unsigned char    Att[2];                // New attenuator setting for each channel (only used
when Cmd == 2)
  unsigned char    Preamp[2];         // New preamp setting for each channel (only used when Cmd == 2)
} TSMW_IQIF_TRIG_CTRL_t;

#endif // _TSMWIQINTERFACETYPES_H
```

A.2 TSMWIQInterfaceFunc.h

```
/**
 * TSMW Project   TSMW IQ Interface DLL
 *
 * @file         TSMWIQInterfaceFunc.h
 * @abstract
 *
 * @copyright     (c) 2007 Rohde & Schwarz GmbH & Co. KG, Munich
 * @author        Markus Herdin, Johannes Dommel
 * @version
 *    06.11.2007  Hd start
 *
 */

#include "TSMWIQInterfaceTypes.h"

#ifndef  _TSMWIQINTERFACEFUNC_H
#define  _TSMWIQINTERFACEFUNC_H

#if defined(__cplusplus) || defined(__cplusplus__)
extern "C"
{
#endif
```

```
//------------------------------------------------------------------------------------------------
/**
 * Get TSMW IQ Interface version.
 *
 * @return int    32-Bit integer representing the 4-byte version code Major.Minor.Patch.QFE
 */
int __stdcall TSMWGetVersion_c ( void );
//------------------------------------------------------------------------------------------------
/**
 * Get TSMW IQ Interface version as text
 *
 * @return char   Version as text
 */
char* __stdcall TSMWGetVersionText_c ( void );
//------------------------------------------------------------------------------------------------
/**
 * Initialize the TSMW IQ Data Interface DLL. Has to be called before any other function is called.
 *
 * @return int    Errorcode, 0 if successful
 */
int __stdcall TSMWInitInterface_c ( void );
//------------------------------------------------------------------------------------------------
/** Release the TSMW IQ Data Interface DLL. Has to be called before the dll is unloaded.
 *
 * @return int    Errorcode, 0 if successful
 */
int __stdcall TSMWReleaseInterface_c ( void );
//------------------------------------------------------------------------------------------------
/**
 * Get error message and error code. Each function returns an error code, which is equal to zero
 * if no error occured. If an error occured, the corresponding error text is saved and can be
 * aquired by this function.
 *
 * Note: The error code is only valid until another function has been called.
 *
 * @param pErrorCode    OUT: Pointer to int which will receive the error code. Error code will be
 *                      zero if successful
 *
 * @return char*        Pointer to zero-terminated string with error message.
 */
char* __stdcall TSMWGetLastError_c ( int *pErrorCode );
//------------------------------------------------------------------------------------------------
/**
 * Connect to TSMW with given IP address
 *
 * @param IPAddress     IN:  IP address as string
 * @param pTSMWMode     IN:  Pointer to TSMW mode structure
 * @param pTSMWID       OUT: Pointer to unsigned short which receives the TSMW ID
 *
 * @return int          Errorcode, 0 if successful
 */
int __stdcall TSMWConnect_c (char* IPAddress, TSMW_IQIF_MODE_t *pTSMWMode, unsigned short *pTSMWID);
//------------------------------------------------------------------------------------------------
/**
 * Addes/modifies filter specifications for an existing measurement session. If a filter specification
 * with the given ID has already been sent to the TSMW, a new call of TSMWIQSetup_c with the same

 * filter ID will overwrite the previous filter specification. This however is not the case if
 * the previous filter specification is used by a different application (with a different
 * IP connection to the TSMW).
 *
 * @param TSMWID          IN: TSMW ID
 * @param pFilterSpec      IN: Pointer to TSMW_IQIF_FILTER_PARAM_t structure containing filter specification.
 * @param plCoeff         IN: Pointer to filter coefficients
 *
 * @return int            Errorcode, 0 if successful
 */
int __stdcall TSMWIQSetup_c (unsigned short TSMWID,
                           TSMW_IQIF_FILTER_PARAM_t *pFilterParam,
                           long *plCoeff);
//-----------------------------------------------------------------------------------------------
/**
 * Request receiver resources.
 *
 * @param TSMWID          IN: TSMW ID
 * @param IsRequest       IN: 1: Request receiver resource
 *                          0: Release receiver resource
 * @param pResourceID      OUT (if resource is requested): receives resource id
 *                       IN  (if resource is released) : specifies resource id to release
 * @param pParam          IN: Resource parameter structure
 *
 * @return int            Errorcode, 0 if successful
 */
int __stdcall TSMWIQResourceRequest_c (unsigned short TSMWID, unsigned long IsRequest,
                                    unsigned long *pResourceID, TSMW_IQIF_RESOURCE_PARAM_t *pParam);
//-----------------------------------------------------------------------------------------------
/**
 * Start a new measurement. Measurements are scheduled on the TSMW. The measurement specification
 * includes a priority and might also include (several) start times at which the measurement can
 * be started. The TSMW schedules automatically the measurement request with the highest priority
 * and the earliest start time. Start times given by the pStartTimes parameter are exclusive, i.e.
 * only a single measurement is scheduled but any of the given start times can be used. If no start
 * times are given, the measurement is scheduled as early as possible.
 *
 * @param TSMWID          IN: ID of TSMW to use for measurement
 * @param pMeasRequestID   OUT: Pointer to MeasRequestID variable, will receive the measurement
 *                       request ID.
 * @param pStartTimes      IN: Pointer to array with start times (optional), if not used: NULL
 * @param NoOfStartTimes   IN: Number of start times given, if no start time given: 0
 * @param pMEAS_CTRL       IN: Measurement control structure
 * @param pCHANNEL_CTRL1   IN: Channel control structure for RF channel 1, NULL if RF channel 1 is
 *                       not used by this meas. req.
 * @param pCHANNEL_CTRL2   IN: Channel control structure for RF channel 2, NULL if RF channel 2 is
 *                       not used by this meas.req.
 *
 * @return int            Errorcode, 0 if successful
 */
int __stdcall TSMWIQMeasure_c (unsigned short TSMWID, unsigned long *pMeasRequestID,
                             unsigned __int64 *pStartTimes, long NoOfStartTimes,
                             TSMW_IQIF_MEAS_CTRL_t *pMEAS_CTRL,
                             TSMW_IQIF_CH_CTRL_t *pCHANNEL_CTRL1,
                             TSMW_IQIF_CH_CTRL_t *pCHANNEL_CTRL2 );
//-----------------------------------------------------------------------------------------------
/**
 * Start an extended measurement. Similar to TSMWIQMeasure_c, but allows to specify scheduling
 * parameters and/or to start periodic measurement requests.
 *
 * @param TSMWID
 * @param pMeasRequestID
 * @param pStartTimes
 * @param NoOfStartTimes
 * @param pMEAS_CTRL
 * @param pCHANNEL_CTRL1
 * @param pCHANNEL_CTRL2
 * @param pSchedParam      IN: Scheduling parameter structure. Has to be specified if the
 *                    measurement request shall use a specific receiver resource class
 *                    that was previously allocated via TSMWIQResourceRequest_c.
 *                    If == NULL, no receiver resource class will be used.
 *                    NOTE: Measurements that do not use a specific receiver resource class
 *                    will not be considered for receiver capacity limitation!
 * @param pPeriodParam     IN: Control structure for starting periodic meas.req.s. If == NULL,
 *                    this is a standard measurement, if given, a periodic measurement request
 *                    is given.
 *
 * @return int
 */
int __stdcall TSMWIQMeasureExt_c (unsigned short TSMWID, unsigned long *pMeasRequestID,
                          unsigned __int64 *pStartTimes, long NoOfStartTimes,
                          TSMW_IQIF_MEAS_CTRL_t *pMEAS_CTRL,
                          TSMW_IQIF_CH_CTRL_t *pCHANNEL_CTRL1,
                          TSMW_IQIF_CH_CTRL_t *pCHANNEL_CTRL2,
                          TSMW_IQIF_SCHEDULE_CTRL_t *pSchedParam,
                          TSMW_IQIF_PERIOD_CTRL_t *pPeriodParam );
//-----------------------------------------------------------------------------------------------
// Start a triggered I/Q measurement
int __stdcall TSMWIQMeasureTrig_c (unsigned short TSMWID, unsigned long *pMeasRequestID,
                          unsigned long long *pStartTimes, long NoOfStartTimes,
                          TSMW_IQIF_MEAS_CTRL *pMEAS_CTRL,
                          TSMW_IQIF_CH_CTRL *pCHANNEL_CTRL1,
                          TSMW_IQIF_CH_CTRL *pCHANNEL_CTRL2,
                          TSMW_IQIF_TRIG_CTRL_t *pTriggerParam );
//-----------------------------------------------------------------------------------------------
int __stdcall TSMWIQPeriodCtrl_c (unsigned short TSMWID, TSMW_IQIF_PERIOD_CTRL_t *pParam);
//-----------------------------------------------------------------------------------------------
int __stdcall TSMWIQTriggerCtrl_c (unsigned short TSMWID, TSMW_IQIF_TRIG_CTRL_t *pParam);
//-----------------------------------------------------------------------------------------------
/**
 * Start a streaming measurement. Both "online" and "offline" streaming is possible.
 * "Online" streaming means that stream data is processed online, hence stream data is only buffered
 * in memory (up to the given stream buffer size) until it is aquired via one of the TSMWIQGetStreamXXX
 * functions. As soon as the corresponding data has been aquired, the corresponding memory is automatically free'd.
 *
 * "Offline" streaming means that data is automatically written to a stream data file. After the streaming has
 * finished, the stream data file can be opened and read.
 *
 * Streaming can in general be started with the same parameters as normal (block) measurements. However, bandwidth
 * (data rate) limitations have to be considered for successful streaming without lost data blocks. See TSMW IQ
 * Interface Manual (Option K1).
 *
 * @param TSMWID           IN: ID of TSMW to use for streaming
 * @param pMEAS_CTRL       IN: Measurement control structure
 * @param pCHANNEL_CTRL1   IN: Channel control structure for RF channel 1, NULL if RF channel 1 is
 *                    not used by this meas. req.
 * @param pCHANNEL_CTRL2   IN: Channel control structure for RF channel 2, NULL if RF channel 2 is
 *                    not used by this meas.req.
 * @param pStreamCtrl      IN: Stream control structure
 * @param pFileName        IN: Filename to use for streaming. If omitted (== NULL) online streaming is started.
 *                    During online streaming, stream data blocks up to the specified stream buffer size
 *                    are stored in memory. Data has to be aquired via the TSMWIQGetStreamXXX functions. Data
 *                    handling is automatic, this means that as soon as a stream data block has been read, it is
 *                    automatically free'd.
 *                    If a file name is given, "offline" streaming is started. This means that the stream data
 *                    blocks are automatically stored in a file with the file name <pFileName>_XXX.dat. Here, _XXX
 *                    starts at 000 and counts up. Files are automatically split if they would exceed 2GB file size.
 * @param pDescription     IN: Description that is stored in streaming file
 * @param Flags            IN: Flags:
 *                         0: Do not overwrite existing stream data files
 *                         1: Overwrite existing stream data files
 *
 * @return int             Errorcode, 0 if successful
 */
int __stdcall TSMWIQStream_c (unsigned short TSMWID,
                      TSMW_IQIF_MEAS_CTRL_t *pMEAS_CTRL,
                      TSMW_IQIF_CH_CTRL_t *pCHANNEL_CTRL1,
                      TSMW_IQIF_CH_CTRL_t *pCHANNEL_CTRL2,
                      TSMW_IQIF_STREAM_CTRL_t *pStreamCtrl,
                      char *pFileName, char *pDescription,
                      unsigned int Flags);
//---------------------------------------------------------------------------------------------
/**
 * Stop a streaming measurement
 *
 * @param TSMWID           IN: ID of TSMW to use for streaming
 * @param StreamID         IN: ID of the corresponding stream
 * @param pStreamStatus    IN: Resulting status (see structure description)
 *
 * @return int             Errorcode, 0 if successful
 */
int __stdcall TSMWIQStopStreaming_c (unsigned short TSMWID, unsigned char StreamID,
                           TSMW_IQIF_STREAM_STATUS_t *pStreamStatus);
//---------------------------------------------------------------------------------------------
/**
 * Get current status of streaming measurement.
 *
 * @param StreamID         IN: ID of the corresponding stream
 * @param pStreamStatus    IN: Resulting status (see structure description)
 *
 * @return int             Errorcode, 0 if successful
 */
int __stdcall TSMWIQStreamStatus_c (unsigned char StreamID,
                           TSMW_IQIF_STREAM_STATUS_t *pStreamStatus);
//---------------------------------------------------------------------------------------------
/**
 * Open a stream data file
 *
 * @param pFileName        IN: Filename of the stream data file without _xxx.dat extension
 * @param StreamID         IN: Stream ID to use for working with this stream data file
 * @param pStreamInfo      OUT: Pointer to stream info structure, will receive information about
 *                    recorded stream
 * @param pMeasCtrl        OUT: Pointer to meas.control structure, will receive meas.ctrl structure
 *                    that was used when the stream was started
 * @param pChannelCtrl1    OUT: Pointer to channel ctrl. structure, will receive channel control
 *                    structure that was used when the stream was started. If this RF frontend
 *                    was not used during streaming, the field NoOfChannels will be set to zero
 * @param pChannelCtrl2    OUT: Pointer to channel ctrl. structure for RF channel 2, see pChannelCtrl1
 * @param pFilterSpec      OUT: Pointer to filter specification structure, will receive filter spec.
 *                    that was used during streaming
 * @param pCoeff           OUT: Pointer to coefficient vector, will receive filter coefficients. The maximum
 *                    number of filter coefficients is 3096.
 * @param NoOfCoeffs       IN: Number of filter coefficients that can be stored in pCoeff array
 *
 * @return int             Errorcode, 0 if successful
 */
int __stdcall TSMWIQOpenStreamFile_c (char *pFileName, unsigned char StreamID,
                                      TSMW_IQIF_STREAM_INFO_t *pStreamInfo,
                                      TSMW_IQIF_MEAS_CTRL_t *pMeasCtrl,
                                      TSMW_IQIF_CH_CTRL_t *pChannelCtrl1,
                                      TSMW_IQIF_CH_CTRL_t *pChannelCtrl2,
                                      TSMW_IQIF_FILTER_PARAM_t *pFilterSpec,
                                      long *pCoeff, unsigned long NoOfCoeffs);
//---------------------------------------------------------------------------------------------
/**
 * Close a stream data file.
 *
 * @param StreamID         IN: ID of the corresponding stream
 *
 * @return int             Errorcode, 0 if successful
 */
int __stdcall TSMWIQCloseStreamFile_c (unsigned char StreamID);
//---------------------------------------------------------------------------------------------
/**
 * Check how many measurement results are available.
 *
 * @param pNoOfIQResults   OUT: Pointer to long which receives the number of IQ measurement results
 *                         available
 *
 * @return int             Errorcode, 0 if successful
 */
int __stdcall TSMWIQDataAvailable_c (long *pNoOfIQResults);
//---------------------------------------------------------------------------------------------
/**
 * Get measurement result parameters.
 * NOTE: This function will not delete the corresponding measurement result. This means that it can be called
 * before one of the TSMWIQGetDataXXX functions is called. The "Get" functions, however, will delete the measurement
 * result.
 *
 * @param MeasRequestID    IN: Measurement request ID for which parameters are requested.
 *                         If the measurement result parameters of the next available measurement will be
 *                         delivered.
 * @param TimeOut          IN: Max. timeout in ms to wait for measurement result
 * @param pIQResult        OUT: Pointer to measurement result parameter structure.
 *
 * @return int             Errorcode, 0 if successful
 */
int __stdcall TSMWIQGetResultParam_c (unsigned int MeasRequestID, unsigned int TimeOut, TSMW_IQIF_RESULT_t
*pIQResult);
//---------------------------------------------------------------------------------------------
/**
 * Get measurement result data in Int16 format.
 * NOTE: This function will delete the corresponding measurement result.
 *
 * @param TSMWID           IN: ID of TSMW
 * @param MeasRequestID    IN: Measurement request ID for which parameters are requested.
 *                         If the measurement result parameters of the next available measurement will be
 *                         delivered.
 * @param TimeOut          IN: Max. time in ms to wait for result
 * @param pIQResult        OUT: Pointer to TSMW_IQIF_RESULT structure, will be filled with result parameters,
 *                         see also TSMWIQGetResultParam_c
 * @param pReal            OUT: Pointer to (NoOfSamples x NoOfChannels) short array to receive real part of data.
 *                   The first NoOfSamples elements in pReal[i] will receive the data of channel 1, the next
 *                   NoOfSamples the data of channel 2 etc.
 * @param pImag           OUT: ... imag part
 * @param pScaling        OUT: Pointer to (1 x NoOfChannels) short array to receive
 *                   scaling factor of each channel. This is the reference level in 1/100 dB. I.e. to get the
 *                   complex sample i in dBm one has to calculate (pReal[i] + pImag[i]) * 10^(Scaling/2000).
 * @param pOvfl           OUT: Pointer to (1 x NoOfChannels) short array to receive overflow indicator of each channel
 *                   Is approximately the number of overflows that occured for in the result data (for each channel)
 * @param pCalibrated     OUT: Pointer to (1 x NoOfChannels) short array to receive calibration indicator of each channel
 *                   0: Channel is not calibrated for this setting
 *                   1: Channel is calibrated for this setting
 * @param NoOfSamples     IN: Number of samples reserved in sample buffer, can be smaller than the number of actually
measured
 *                   samples.
 *                   NOTE: Since the measurement result will be deleted when this function is called, the samples
 *                   that were not copied, are lost.
 * @param NoOfChannels    IN: Number of channels reserved in sample buffer, has to be equal to the number of channels that
 *                   were requested in the measurement request
 * @param Reserved1       Has to be 0
 * @param Reserved2       Has to be 0
 *
 * @return int            Errorcode, 0 if successful

 */
int __stdcall TSMWIQGetDataInt16_c (unsigned short TSMWID, unsigned int MeasRequestID,
                              unsigned int TimeOut, TSMW_IQIF_RESULT_t *pIQResult,
                              short* pReal, short* pImag, short* pScaling, unsigned long* pOvfl,
                              unsigned int *pCalibrated,
                              unsigned int NoOfSamples, unsigned int NoOfChannels,
                              int Reserved1, int Reserved2);
//-------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------
/**
 * Get measurement result data in Int32 format.
 * NOTE: This function will delete the corresponding measurement result.
 *
 * See TSMWIQGetDataInt16_c
 */
int __stdcall TSMWIQGetDataInt32_c (unsigned short TSMWID, unsigned int MeasRequestID,
                              unsigned int TimeOut, TSMW_IQIF_RESULT_t *pIQResult,
                              long* pReal, long* pImag, short* pScaling, unsigned long* pOvfl,
                              unsigned int *pCalibrated,
                              unsigned int NoOfSamples, unsigned int NoOfChannels,
                              int Reserved1, int Reserved2);
//-------------------------------------------------------------------------------------------------
/**
 * Get measurement result data in single precision floating point format.
 * NOTE: This function will delete the corresponding measurement result.
 *
 * See TSMWIQGetDataInt16_c
 */
int __stdcall TSMWIQGetDataSingle_c (unsigned short TSMWID, unsigned int MeasRequestID,
                               unsigned int TimeOut, TSMW_IQIF_RESULT_t *pIQResult,
                               float* pReal, float* pImag, short* pScaling, unsigned long* pOvfl,
                               unsigned int *pCalibrated,
                               unsigned int NoOfSamples, unsigned int NoOfChannels,
                               int Reserved1, int Reserved2);
//-------------------------------------------------------------------------------------------------
/**
 * Get measurement result data in double precision floating point format.
 * NOTE: This function will delete the corresponding measurement result.
 *
 * See TSMWIQGetDataInt16_c
 */
int __stdcall TSMWIQGetDataDouble_c (unsigned short TSMWID, unsigned int MeasRequestID,
                              unsigned int TimeOut, TSMW_IQIF_RESULT_t *pIQResult,
                              double* pReal, double* pImag, short* pScaling, unsigned long* pOvfl,
                              unsigned int *pCalibrated,
                              unsigned int NoOfSamples, unsigned int NoOfChannels,
                              int Reserved1, int Reserved2);
//-------------------------------------------------------------------------------------------------
/**
 * Get measurement result data in interleaved single precision floating point format.
 * This means that the I and Q components are copied to pCplxData in interleaved fashion.
 * Interleaving is such that even index elements pCplxData[i] (i starting with zero) will
 * receive the real part and odd index elements the imaginary part. The ordering of channels
 * is the same is for TSMWIQGetDataInt16_c.
 *
 * This means that (2 x NoOfSamples x NoOfChannels) elements have to be received in pCplxData.
 *
 * NOTE: This function will delete the corresponding measurement result.
 *
 * See TSMWIQGetDataInt16_c
 */
int __stdcall TSMWIQGetDataSingleIlv_c (unsigned short TSMWID, unsigned int MeasRequestID,
                              unsigned int TimeOut, TSMW_IQIF_RESULT_t *pIQResult,
                              float* pCplxData, short* pScaling, unsigned long* pOvfl,
                              unsigned int *pCalibrated,
                              unsigned int NoOfSamples, unsigned int NoOfChannels,
                              int Reserved1, int Reserved2);
//-------------------------------------------------------------------------------------------------
/**
 * Get measurement result data in interleaved double precision floating point format.
 * NOTE: This function will delete the corresponding measurement result.
 *
 * See TSMWIQGetDataSingleIlv_c and TSMWIQGetDataInt16_c
 */
int __stdcall TSMWIQGetDataDoubleIlv_c (unsigned short TSMWID, unsigned int MeasRequestID,
                              unsigned int TimeOut, TSMW_IQIF_RESULT_t *pIQResult,
                              double* pCplxData, short* pScaling, unsigned long* pOvfl,
                              unsigned int *pCalibrated,
                              unsigned int NoOfSamples, unsigned int NoOfChannels,
                              int Reserved1, int Reserved2);
//-------------------------------------------------------------------------------------------------
/**
 * Get stream data in single precision floating point format. This functions works for both
 * "online" streaming and "offline" streaming. "Online" streaming means processing stream data
 * on the fly, hence no automatic writing to a stream data file. "Offline" streaming means that
 * data is recorded on the disc in a stream data file. After the streaming has been finished, the
 * stream data file can be opened and data can be read from the file using this function.
 *
 * @param StreamID        IN: ID of the corresponding stream
 * @param TimeOut         IN: Max. time in ms to wait for result
 * @param pIQResult       OUT: Pointer to TSMW_IQIF_RESULT structure, will be filled with result parameters,
 *                   see also TSMWIQGetResultParam_c
 * @param pReal           OUT: Pointer to (NoOfSamples x NoOfChannels) float array to receive real part of data.
 *                   The first NoOfSamples elements in pReal[i] will receive the data of channel 1, the next
 *                   NoOfSamples the data of channel 2 etc.
 * @param pImag           OUT: ... imag part
 * @param pScaling        OUT: Pointer to (1 x NoOfChannels) short array to receive
 *                      scaling factor of each channel. This is the reference level in 1/100 dB. I.e. to get the
 *                      complex sample i in dBm one has to calculate (pReal[i] + pImag[i]) * 10^(Scaling/2000).
 * @param pOvfl            OUT: Pointer to (1 x NoOfChannels) short array to receive overflow indicator of each channel
 *                      Is approximately the number of overflows that occured for in the result data (for each channel)
 * @param pCalibrated      OUT: Pointer to (1 x NoOfChannels) short array to receive calibration indicator of each channel
 *                      0: Channel is not calibrated for this setting
 *                      1: Channel is calibrated for this setting
 * @param Offset           Only used during "offline" streaming, i.e. reading from a recorded stream data file.
 *                      This is the sample index offset at which to start to get data. 0 means start at the very first
 *                      sample that was recorded.
 * @param NoOfSamples      IN: Number of samples to copy
 * @param NoOfChannels     IN: Number of channels reserved in sample buffer, has to be equal to the number of channels that
 *                      were requested in the measurement request
 *
 * @return int            Errorcode, 0 if successful
 */
int __stdcall TSMWIQGetStreamSingle_c (unsigned char StreamID, unsigned int TimeOut, TSMW_IQIF_RESULT_t *pIQResult,
                                  float* pReal, float* pImag, short* pScaling, unsigned long* pOvfl,
                                  unsigned int *pCalibrated, unsigned __int64 Offset,
                                  unsigned int NoOfSamples, unsigned int NoOfChannels);
//---------------------------------------------------------------------------------------------
/**
 * Get streaming data block in double format.
 *
 * See TSMWIQGetStreamSingle_c and TSMWIQGetDataInt16_c.
 */
int __stdcall TSMWIQGetStreamDouble_c (unsigned char StreamID, unsigned int TimeOut, TSMW_IQIF_RESULT_t *pIQResult,
                                  double* pReal, double* pImag, short* pScaling, unsigned long* pOvfl,
                                  unsigned int *pCalibrated, unsigned __int64 Offset,
                                  unsigned int NoOfSamples, unsigned int NoOfChannels);
//---------------------------------------------------------------------------------------------
/**
 * Get streaming data block in interleaved single (float) format.
 *
 * See TSMWIQGetStreamSingle_c, TSMWIQGetDataSingleIlv_c and TSMWIQGetDataInt16_c.
 */
int __stdcall TSMWIQGetStreamSingleIlv_c (unsigned char StreamID, unsigned int TimeOut, TSMW_IQIF_RESULT_t *pIQResult,
                                    float* pCplxData, short* pScaling, unsigned long* pOvfl,
                                    unsigned int *pCalibrated, unsigned __int64 Offset,
                                    unsigned int NoOfSamples, unsigned int NoOfChannels);
//---------------------------------------------------------------------------------------------
//
/**
 * Get streaming data block in interleaved double format.
 *
 * See TSMWIQGetStreamSingle_c, TSMWIQGetDataSingleIlv_c and TSMWIQGetDataInt16_c.
 */
int __stdcall TSMWIQGetStreamDoubleIlv_c (unsigned char StreamID, unsigned int TimeOut, TSMW_IQIF_RESULT_t *pIQResult,
                                    double* pCplxData, short* pScaling, unsigned long* pOvfl,
                                    unsigned int *pCalibrated, unsigned __int64 Offset,
                                    unsigned int NoOfSamples, unsigned int NoOfChannels);
//---------------------------------------------------------------------------------------------
/**
 * Shutdown TSMW
 */
int __stdcall TSMWShutdown_c (unsigned short TSMWID);
//---------------------------------------------------------------------------------------------
/**
 * Enable tracking generator for selected frontends.
 *
 * @param TSMWID          IN: ID of TSMW to use for streaming
 * @param Frontends       IN: Selected frontends (tracking generator, i.e. test out is only possible
 *                         for frontend 1):
 *                           0: Test out disabled
 *                           1: Test out for frontend 1 enabled
 *
 * @return int            Errorcode, 0 if successful
 */
int __stdcall TSMWTGEnable_c (unsigned short TSMWID, unsigned int Frontends);
//---------------------------------------------------------------------------------------------
/**
 * Enable external reference.
 * NOTE: One has to ensure that the external reference is already connected when calling this function
 * to enable usage of the external reference. If the external reference has been enabled but there is

 * no external reference clock connected to the TSMW, measurements on rf channel 2 will not work.
 *
 * @param TSMWID          IN: ID of TSMW to use for streaming
 * @param Enable          IN: 0: Enable external reference
 *                         1: Disable external reference
 *
 * @return int            Errorcode, 0 if successful
 */
int __stdcall TSMWExtRefEnable_c (unsigned short TSMWID, int Enable);
//---------------------------------------------------------------------------------------------
/**
 * Set reference oscillator DAC for frequency correction. Only possible if GPS synchronization
 * has been disabled.
 *
 * @param TSMWID          IN: ID of TSMW to use for streaming
 * @param Value           IN: DAC value to set (0..1023, sensible range is 200 - 800)
 *
 * @return int            Errorcode, 0 if successful
 */
int __stdcall TSMWSetDAC_c (unsigned short TSMWID, short Value);
//---------------------------------------------------------------------------------------------
/**
 * Enable/disable GPS synchronization. Default: enabled.
 *
 * @param TSMWID          IN: ID of TSMW to use
 * @param Enable          IN: TRUE -> Enable, FALSE -> Disable
 *
 * @return int            Errorcode, 0 if successful
 */
int __stdcall TSMWGPSSync_c (unsigned short TSMWID, int Enable);
//---------------------------------------------------------------------------------------------
/**
 * Enable/disable GPS data subscription. NMEA and UBX GPS data lines will be received from the TSMW and
 * are stored in a ring buffer of the given size. If the buffer size is exceeded, the oldest
 * received NMEA string or UBX message is overwritten. The NMEA and UBX buffers are independent.
 *
 * @param TSMWID          IN: ID of TSMW to use
 * @param NoOfBufferedLines  IN: Number of buffered NMEA lines
 * @param Enable          IN: TRUE -> Enable, FALSE -> Disable
 *
 * @return int            Errorcode, 0 if successful
 */
int __stdcall TSMWGPSEnable_c (unsigned short TSMWID, int NoOfBufferedLines, int Enable);
//---------------------------------------------------------------------------------------------
/**
```

```
 * Send NMEA command string to GPS
 *
 * @param TSMWID          IN: ID of TSMW to use
 * @param pNMEAString      IN: NMEA string to send to TSMW
 *
 * @return int            Errorcode, 0 if successful
 */
int __stdcall TSMWGPSSendNMEACmd_c(int TSMWID, char* pNMEAString);
//--------------------------------------------------------------------------------------------
/**
 * Send UBX message to GPS
 *
 * @param TSMWID          IN: ID of TSMW to use
 * @param pUBXMsg          IN: UBX message to send to TSMW
 * @param UBXMsgLength     IN: UBX message length
 *
 * @return int            Errorcode, 0 if successful
 */

int __stdcall TSMWGPSSendUBXCmd_c(int TSMWID, void* pUBXMsg, unsigned int UBXMsgLength);
//--------------------------------------------------------------------------------------------
/**
 * Clear GPS NMEA and UBX buffer.
 *
 * @return int            Errorcode, 0 if successful
 */
int __stdcall TSMWGPSClearBuffer_c ( void );
//--------------------------------------------------------------------------------------------
/**
 * Get next available GPS NMEA data line. Returns an empty line if no NMEA strings is available.
 * This function can be called from another thread than all the other functions. However, it is
 * not possible to call this function from more than one thread!
 *
 * @return char*           Pointer to NMEA text string, valid until next call of function.
 */
char* __stdcall TSMWGPSGetNMEALine_c ();
//--------------------------------------------------------------------------------------------
/**
 * Get next available GPS UBX message. Returns NULL if no UBX message available. This function
 * can be called from another thread than all the other functions. However, it is not possible
 * to call this function from more than one thread!
 *
 * @param pLength          OUT: Pointer to unsigned int to receive length of UBX message
 *
 * @return void*           Pointer to UBX message, valid until next call of function
 */
void* __stdcall TSMWGPSGetUBXMsg_c (unsigned int *pLength);
//--------------------------------------------------------------------------------------------
/**
 * Get estimate of current IQ-time at TSMW (Current IQ-counter value). The IQ-counter value
 * runs at TSMW sample rate (21.9444MHz). The IQ-time has 48 valid bits. The IQ-time is the time
 * that is used to start measurements.
 *
 * @param TSMWID          IN: ID of TSMW to use for streaming
 * @param pIQTime          OUT: Pointer to __int64 to receive current IQ time.
 *
 * @return int            Errorcode, 0 if successful
 */
int __stdcall TSMWGetIQTime_c (unsigned short TSMWID, unsigned __int64 *pIQTime);
//--------------------------------------------------------------------------------------------
```

```
/**
 * Set/read TSMW trigger IOs
 *
 * @param TSMWID          IN: ID of TSMW to use for streaming
 * @param TriggerLine     Trigger line that is addressed
 *                        1: Trigger 1
 *                        2: Trigger 2
 * @param Out             0: Use as input
 *                      1: Use as output
 * @param pTrig1Value     IN/OUT: Set/get current trigger level for trigger line 1 (0: low, 1: high)
 * @param pTrig2Value     IN/OUT: Set/get current trigger level for trigger line 2 (0: low, 1: high)
 *
 * @return int            Errorcode, 0 if successful
 */
int __stdcall TSMWTrigger_c (unsigned short TSMWID, int TriggerLine, int Out,
                        int *pTrig1Value, int *pTrig2Value);

#if defined(__cplusplus) || defined(__cplusplus__)
}
#endif

#endif // _TSMWIQINTERFACEFUNC_H
```

B M-Functions

All R&S TSMW-K1 functions for the R&S TSMW MATLAB IQ interface are explained in the following sub chapters. Additionally the corresponding C++ functions with the correct command syntax are mention. Most MATLAB parameters correspond to the C++ parameters.

B.1 TSMWInitInterface

Description:

Load the R&S TSMW IQ interface library `TSMWIQInterface.dll`. The function `TSMWInitInterface` has to be called once before any other action can be performed on the TSMW.

Command syntax:

```
[ErrorText, ErrorCode] = TSMWInitInterface;
```

Parameter(s):

-/-

Return value(s):

`ErrorText`	If no error occurs an empty string is returned. Otherwise the message text of the error is returned.
`ErrorCode`	0 if successful. Otherwise the error code is returned.

Corresponding C++ command:

```
[int ErrorCode = TSMWInitInterface_c();
```

B.2 TSMWReleaseInterface

Description:

Unload the external R&S TSMW IQ interface library and disconnect from R&S TSMW(s).

MATLAB Interface Library

To secure your MATLAB application from crash you have to call the function TSMWReleaseInterface before you can close the application.

Command syntax:

```
[ErrorCode] = TSMWReleaseInterface;
```

Parameter(s):

-/-

Return value(s):

`ErrorCode`	0 if successful. Otherwise the error code is returned.

Corresponding C++ command:

```
int ErrorCode = TSMWReleaseInterface_c();
```

B.3 TSMWConnect

Description:

Establish a connection to the TSMW with the given IP address and given options.

Command syntax:

```
[ErrorCode, TSMWID] = TSMWConnect(IPAddress, TSMWOptions);
```

Parameter(s):

`IPAddress`	IP address of the R&S TSMW.
`TSMWOptions`	Specifies R&S TSMW operating options. If omitted, default options will be used. That means both frontends and all amplifiers are enabled. **Note:** When the R&S TSMW is powered, the current firmware version always turns the frontend on.

Return value(s):

`ErrorCode`	0 if successful. Otherwise the error code is returned.
`TSMWID`	The ID represents the R&S TSMW connection.

Corresponding C++ command:

```
int ErrorCode = TSMWConnect_c(char* IPAddress,
                              TSMW_IQIF_MODE_t *pTSMWMode,
                              unsigned short *pTSMWID);
```

B.4 TSMWIQSetup

Description:

Transmit a filter specification to the R&S TSMW. The field `FilterID` in the structure `FilterSpec` contains the ID of the filter.

Overwrite an existing filter with the same ID

To overwrite an existing filter with a new transmitted filer with the same ID, the previous filter specification has to be used with the same application with the same IP connection to the R&S TSMW.

Resampling filter

In order to perform resampling on the R&S TSMW, an appropriate resampling filter has to be transmitted to the R&S TSMW. For filter design you might use the provided R&S TSMW Filter Design Tool.

Command syntax:

```
[ErrorCode] = TSMWSetup(TSMWID, FilterSpec);
```

Parameter(s):

`TSMWID`	Specifies the R&S TSMW ID to use for measurement.
`FilterSpec`	Filter specification structure. For a detailed description of the structure refer to chapter Structure: FilterSpec on page 32 and to Filter Design Dialog on page 43.

Return value(s):

`ErrorCode`	0 if successful. Otherwise the error code is returned.

Corresponding C++ command:

```
[ErrorCode] = TSMWIQSetup_c(unsigned short TSMWID,
                  TSMW_IQIF_FILTER_PARAM_t *pFilterParam,
                                          long *plCoeff);
```

B.5 TSMWGetLastError

Description:

Each function returns an error code: 0 if successful. Otherwise a number which defines the error code. To get the corresponding error text calls the function `TSMWGetLastError`. The function returns the error information of the occurred error and reset the error code variable again to zero.

Variable ErrorCode

Please keep in mind that only one ErrorCode variable is used for all functions. Therefore only the current returned error code is available.

Command syntax:

```
[ErrorText, ErrorCode] = TSMWGetLastError;
```

Parameter(s):

-/-

Return value(s):

`ErrorCode`	0 if successful. Otherwise the error code is returned.
`ErrorText`	If no error message exists the value zero is returned

Corresponding C++ command:

```
char* ErrorText = TSMWGetLastError_c( int *pErrorCode);
```

B.6 TSMWIQMeasure

Description:

The function starts a new measurement with given parameters. The measurement parameters are defined in the structure `MeasCtrl`.

Measurements are scheduled on the TSMW. The measurement specification includes a priority and might also include (several) start times at which the measurement can be started. The TSMW schedules automatically the measurement request with the highest priority and the earliest start time. Start times given by the StartTimes parameter are exclusive, i.e. only a single measurement is scheduled but any of the given start times can be used. If no start times are given, the measurement is scheduled as early as possible.

Extend measurement

Via the measurement control structure it is possible to start single measurement or streaming measurement. Furthermore following attributes of the measurement can be set:

- schedules the measurement
- defines the measurement as periodic task
- define a trigger when to start and stop the measurement.

Command syntax:

```
[ErrorCode, MeasRequestID] = TSMWMeasure(TSMWID,
                                          StartTimes,MeasCtrl);
```

Parameter(s):

`TSMWID`	Specifies the TSMW to use for measurement.
`StartTimes`	Defines one or more start time(s) at which the measurement can be started. It is a multiple of 1/(395e6 / 18) seconds (approximately 45ns) and relative to start-up of the TSMW. If the parameter is empty, the measurement starts as soon as possible. For the current I/Q time on the R&S TSMW an estimate can be obtained by calling TSMWGetIQTime.
`MeasCtrl`	Measurement control structure. A template of the measurement control structure is provided by the MATLAB scripts “MeasCtrlTemplate”, “MeasCtrlTemplate_RF1” and “MeasCtrlTemplate_RF2”. If the field “ChannelCtrl1” is present in the MeasCtrl structure, a measurement on RF-channel 1 is performed. If the field “ChannelCtrl2” is present, a measurement on RF-channel 2 is performed. A measurement on both channels is performed if both fields are present. For further information about the fields refer to chapter Structure: MeasCtrl on page 24 and provided MATLAB example application.

Return value(s):

`ErrorCode`	0 if successful. Otherwise the error code is returned.
`MeasRequestID`	ID for this measurement request.

Corresponding C++ command:

```
int ErrorCode = TSMWIQMeasure_c(
                      unsigned short TSMWID,
                      unsigned long *pMeasRequestID,
                      unsigned __int64 *pStartTimes,
                      long NoOfStartTimes,
                      TSMW_IQIF_MEAS_CTRL_t *pMEAS_CTRL,
                      TSMW_IQIF_CH_CTRL_t *pCHANNEL_CTRL1,
                      TSMW_IQIF_CH_CTRL_t *pCHANNEL_CTRL2 );
int ErrorCode = TSMWIQMeasureExt_c (
                      unsigned short TSMWID,
                      unsigned long *pMeasRequestID,
                      unsigned __int64 *pStartTimes,
                      long NoOfStartTimes,
                      TSMW_IQIF_MEAS_CTRL_t *pMEAS_CTRL,
                      TSMW_IQIF_CH_CTRL_t *pCHANNEL_CTRL1,
                      TSMW_IQIF_CH_CTRL_t *pCHANNEL_CTRL2,
                    TSMW_IQIF_SCHEDULE_CTRL_t *pSchedParam,
                    TSMW_IQIF_PERIOD_CTRL_t *pPeriodParam );
int ErrorCode = TSMWIQMeasureTrig_c (
                      unsigned short TSMWID,
                      unsigned long *pMeasRequestID,
                      unsigned long long *pStartTimes,
                      long NoOfStartTimes,
                      TSMW_IQIF_MEAS_CTRL *pMEAS_CTRL,
                      TSMW_IQIF_CH_CTRL *pCHANNEL_CTRL1,
                      TSMW_IQIF_CH_CTRL *pCHANNEL_CTRL2,
                     TSMW_IQIF_TRIG_CTRL_t *pTriggerParam );
```

B.7 TSMWResourceRequest

Description:

The function requests a receiver resource for a measurement task. Therefore enter the resourceID of the resource. The value will be entered automatically into the m-structure field `MeasCtrlSchedCtrl`.

If too many requests are sent to the same resource, the system schedule all waiting measurement tasks automatically. The schedule algorithm manages the access to the resource according to the priority number (m-structure field `MeasCtrl.MeasCtrl.Priority`).

The given priority number increase dynamically according to the waiting time (observation time).

If a measurement task does not have a priority value at creation time, the system will attach the lowest possible priority value to it => 0.

Command syntax:

```
[ErrorCode, ResourceID] = TSMWResourceRequest(TSMWID,
                          IsRequest, ResourceID, ResourceParam);
```

Parameter(s):

`TSMWID`	Specifies the TSMW to use.
`IsRequest`	1: Request receiver resource. 0: Release receiver resource.
`ResourceID`	Only relevant for IsRequest == 0. Resource ID of receiver resource that shall be released.
`ResourceParam`	Resource parameter structure. For further information about the fields refer to chapter Structure: MeasCtrl on page 24.

Return value(s):

`ErrorCode`	0 if successful. Otherwise the error code is returned.
`ResourceID`	Only relevant for IsRequest == 0 Returns the resource ID if resource was granted.

Corresponding C++ command:

```
int ErrorCode = TSMWIQResourceRequest_c (
                       unsigned short TSMWID,
                       unsigned long IsRequest,
                       unsigned long *pResourceID,
                       TSMW_IQIF_RESOURCE_PARAM_t *pParam);
```

B.8 TSMWIQPeriodCtrl

Description:

The function set and modify the parameters of a periodic measurement request.

Command syntax:

```
[ErrorCode] = TSMWIQPeriodCtrl(TSMWID, PeriodCtrl);
```

Parameter(s):

`TSMWID`	Specifies the TSMW to use.
`PeriodCtrl`	Control structure of the periodic measurement requests. For further information about the fields refer to chapter Structure: MeasCtrl on page 24.

Return value(s):

`ErrorCode`	0 if successful. Otherwise the error code is returned.

Corresponding C++ command:

```
int ErrorCode = TSMWIQPeriodCtrl_c (
                         unsigned short TSMWID,
                         TSMW_IQIF_PERIOD_CTRL_t *pParam);
```

B.9 TSMWIQDataAvailable

Description:

The function returns the number of the available measurement results. The value zero is returned when no data is available.

Command syntax:

```
[ErrorCode, NoOfDataBlocks] = TSMWIQDataAvailable;
```

Parameter(s):

`-/-`

Return value(s):

`ErrorCode`	0 if successful. Otherwise the error code is returned.
`NoOfDataBlocks`	Specifies the number of available data blocks.

Corresponding C++ command:

```
int ErrorCode = TSMWIQDataAvailable_c(long *pNoOfIQResults);
```

B.10 TSMWIQGetResultParam

Description:

This function returns the parameters of a measurement result.

Do not delete measurement result

The function will not delete the measurement result. This means that it can be called before one of the TSMWIQGetDataXXX functions is called. The "Get" functions, however, will delete the measurement result.

Command syntax:

```
[ErrorCode, IQResultParam] = TSMWIQDGetResultParam(
                             MeasRequestID, TimeOut);
```

Parameter(s):

`MeasRequestID`	Specifies the measurement request ID of the measurement result to wait for. If 0, get parameters of next available measurement data block (with lowest measurement request ID).
`TimeOut`	Defines the time in milliseconds to wait for the measurement result.

Return value(s):

`ErrorCode`	0 if successful. Otherwise the error code is returned.
`IQResultParam`	Measurement result parameters. For information about the fields refer to chapter Structure: TSMWIQResult on page 33.

Corresponding C++ command:

```
int ErrorCode = TSMWIQGetResultParam_c(
                             unsigned int MeasRequestID,
                             unsigned int TimeOut,
                             TSMW_IQIF_RESULT_t *pIQResult);
```

B.11 TSMWIQGetData

Description:

This function retrieves I/Q data in complex double array format.

Delete measurement result

The function will delete the corresponding measurement result on the R&S TSMW.

Command syntax:

```
[ErrorCode,IQResultParam,
IQData,Overflow,
   Calibrated] = TSMWIQGetData(TSMWID,MeasRequestID,TimeOut,
                               NoOfSamples,NoOfChannels);
```

Parameter(s):

`TSMWID`	Defines the used TSMW.
`MeasRequestID`	Specifies the measurement request ID of the measurement result to wait for. If 0, get parameters of next available measurement data block (with lowest measurement request ID).
`TimeOut`	Defines the time in milliseconds to wait for the measurement response result.
`NoOfSamples`	Defines the number of expected samples. Only the specified number of samples will be copied from the measurement result. **Note:** The no. of requested samples has to be a multiple of 8.
`NoOfChannels`	Specifies the number of expected channels. This has to be equal to the total number of channels the corresponding measurement request covers. I.e.: Measuring only on frontend 1 with only 1 sub-channel means NoOfChannels = 1. Measuring on frontend 1 and 2 with 1 sub-channel active on frontend 1 and 2 sub-channels on frontend 2 mean NoOfChannels = 3.

Return value(s):

`ErrorCode`	0 if successful. Otherwise the error code is returned.
`IQResultParam`	Measurement result parameters. For information about the fields refer to chapter Structure: TSMWIQResult on page 33.

`IQData`	Complex I/Q data array (already scaled with reference level 1dBm).
Overflow	Array of `1 x NoOfChannels` The array contains for each channel the number of overflows that have occurred during the measurement
`Calibrated`	Array of `1 x NoOfChannels` The array contains for each channel the calibration information. 0 if corresponding measurement setting is not calibrated. Otherwise 1.

Corresponding C++ command:

The parameters `Reserved1` and `Reserved2` have to be zero.

```
int ErrorCode = TSMWIQGetDataDouble_c(
                unsigned short TSMWID,
                unsigned int MeasRequestID,
                unsigned int TimeOut,
                TSMW_IQ_RESULT *pIQResult,
                short* pReal, short* pImag, short* pScaling,
                unsigned long* pOvfl,
                unsigned int *pCalibrated,
                unsigned int NoOfSamples,
                unsigned int NoOfChannels,
                int Reserved1,int Reserved2);
```

Additional information for specific C++ return values:

pReal: Pointer to a short array with reserved elements for NoOfSamples x NoOfChannels The first NoOfSamples data samples correspond to the first sub channel of the first frontend. The next NoOfSamples data samples correspond to the next sub channel (if applicable) of the first frontend etc..

pImag: Quadrature data array. Ordering as in pReal.

pScaling: Pointer to a short array with NoOfChannels reserved elements. The user has to reserve the space for the array, before. The array receives for each channel the reference level of the data in 1/100 dBm.

pOvfl: : Pointer to a short array with NoOfChannels reserved elements. The user has to reserve the space for the array, before. The array receives for each channel the information if an overflow has occurred during the measurement. 0 if no overflow occurs. Otherwise 1.

pCalibration: Pointer to a short array with NoOfChannels reserved elements. The array receives for each channel the calibration information. 0 if corresponding measurement setting is not calibrated. Otherwise 1.

B.12 TSMWIQGetDataInt16

Description:

This function unpacks the measurement result into 16 bit integer array format.

Delete measurement result

The function will delete the corresponding measurement result.

Command syntax:

```
[ErrorCode,
IQResultParam,
IData,QData,Scaling,
Overflow,Calibrated] = TSMWIQGetDataInt16(TSMWID,
                                          MeasRequestID,TimeOut,
                                          NoOfSamples,NoOfChannels);
```

Parameter(s):

`TSMWID`	Defines the used TSMW.
`MeasRequestID`	Specifies the measurement request ID of the measurement result to wait for. If 0, get parameters of next available measurement data block (with lowest measurement request ID).
`TimeOut`	Defines the time in milliseconds to wait for the measurement response result.
`NoOfSamples`	Defines the number of expected samples. Only the specified number of samples will be copied from the measurement result. **Note:** For the functions TSMWIQGetDataInt16 and TSMWIQGetDataInt32, the number of requested samples has to be a multiple of 8. This is not the case for functions the deliver single or double precision data (e.g. TSMWIQGetDataDouble, TSMWIQGetStreamDouble, ...).
`NoOfChannels`	Specifies the number of expected channels. This has to be equal to the total number of channels the corresponding measurement request covers. I.e.: Measuring only on frontend 1 with only 1 sub-channel means NoOfChannels = 1. Measuring on frontend 1 and 2 with 1 sub-channel active on frontend 1 and 2 sub-channels on frontend 2 mean NoOfChannels = 3.

Return value(s):

`ErrorCode`	0 if successful. Otherwise the error code is returned.
`IQResultParam`	Measurement result parameters. For information about the fields refer to chapter Structure: TSMWIQResult on page 33 defined.
`IData`	`NoOfSamples x NoOfChannels` in-phase data array
`QData`	Quadrature data array. Ordering as in IData.
`Scaling`	Array `of 1 x NoOfChannels` The array contains for each channel the reference level of the data in 1/100 dBm.
`Overflow`	Array of `1 x NoOfChannels` The array contains for each channel the number of overflows that have occurred during the measurement.
`Calibrated`	Array of `1 x NoOfChannels` The array contains for each channel the calibration information. 0 if corresponding measurement setting is not calibrated. Otherwise 1.

Corresponding C++ command:

The parameters `Reserved1` and `Reserved2` have to be zero.

```
int ErrorCode = TSMWIQGetDataInt16_c(
                unsigned short TSMWID,
                unsigned int MeasRequestID,
                unsigned int TimeOut,
                TSMW_IQIF_RESULT_t *pIQResult,
                short* pReal, short* pImag, short* pScaling,
                unsigned long* pOvfl,
                unsigned int *pCalibrated,
                unsigned int NoOfSamples,
                unsigned int NoOfChannels,
                int Reserved1, int Reserved2);
```

Additional information for specific C++ return values:

pReal: Pointer to a short array with reserved elements for `NoOfSamples x NoOfChannels` The first NoOfSamples data samples correspond to the first sub channel of the first frontend. The next `NoOfSamples` data samples correspond to the next sub channel (if applicable) of the first frontend etc..

pImag: Quadrature data array. Ordering as in `pReal`.

pScaling: Pointer to a short array with `NoOfChannels` reserved elements. The user has to reserve the space for the array, before. The array receives for each channel the reference level of the data in `1/100` dBm.

pOvfl: : Pointer to a short array with `NoOfChannels` reserved elements. The user has to reserve the space for the array, before. The array receives for each channel the information if an overflow has occurred during the measurement. 0 if no overflow occurs. Otherwise 1.

pCalibration: Pointer to a short array with `NoOfChannels` reserved elements. The array receives for each channel the calibration information. 0 if corresponding measurement setting is not calibrated. Otherwise 1.

B.13 TSMWIQGetDataInt32

Description:

This function unpacks the measurement result into 32 bit integer array format.

Delete measurement result

The function will delete the corresponding measurement result.

See also TSMWIQGetDataInt16.

Command syntax:

```
[ErrorCode,
IQResultParam,
IData,QData,Scaling,
Overflow,Calibrated] = TSMWIQGetDataInt32(TSMWID,
                                       MeasRequestID,TimeOut,
                                       NoOfSamples,NoOfChannels);
```

Parameter(s):

See TSMWIQGetDataInt16.

Return value(s):

See TSMWIQGetDataInt16.

Corresponding C++ command:

The parameters `Reserved1` and `Reserved2` have to be zero.

```
int ErrorCode = TSMWIQGetDataInt32_c(
                     unsigned short TSMWID,
                     unsigned int MeasRequestID,
                     unsigned int TimeOut,
                     TSMW_IQ_RESULT_t *pIQResult,
                     short* pReal, short* pImag,short* pScaling,
                     unsigned long* pOvfl,
                     unsigned int *pCalibrated,
                     unsigned int NoOfSamples,
                     unsigned int NoOfChannels,
                     int Reserved1, int Reserved2);
```

B.14 TSMWIQGetDataSingle

Description:

This function unpacks the measurement result into single precision floating point array format.

Delete measurement result

The function will delete the corresponding measurement result.

See also TSMWIQGetDataInt16.

Command syntax:

```
[ErrorCode,
IQResultParam,
IData,QData,Scaling,
Overflow,Calibrated] = TSMWIQGetDataSingle(TSMWID,
                                      MeasRequestID,TimeOut,
                                   NoOfSamples,NoOfChannels);
```

Parameter(s):

See TSMWIQGetDataInt16.

Return value(s):

See TSMWIQGetDataInt16.

Corresponding C++ command:

The parameters `Reserved1` and `Reserved2` have to be zero.

```
int ErrorCode = TSMWIQGetDataSingle_c(
                 unsigned short TSMWID,
                 unsigned int MeasRequestID,
                 unsigned int TimeOut,
                 TSMW_IQ_RESULT_t *pIQResult,
                 short* pReal, short* pImag,short* pScaling,
                 unsigned long* pOvfl,
                 unsigned int *pCalibrated,
                 unsigned int NoOfSamples,
                 unsigned int NoOfChannels,
                 int Reserved1, int Reserved2);
```

B.15 TSMWIQGetDataDouble

Description:

This function unpacks the streaming measurement result into double precision floating point array format.

Delete measurement result

The function will delete the corresponding measurement result.

See also TSMWIQGetDataInt16.

Command syntax:

```
[ErrorCode,IQResultParam,
IData,QData,Scaling,
Overflow,Calibrated] = TSMWIQGetDataDouble(TSMWID,
                            MeasRequestID,TimeOut,
                            NoOfSamples,NoOfChannels);
```

Parameter(s):

See TSMWIQGetDataInt16.

Return value(s):

See TSMWIQGetDataInt16.

Corresponding C++ command:

The parameters `Reserved1` and `Reserved2` have to be zero.

```
int ErrorCode = TSMWIQGetDataDouble_c(
                 unsigned short TSMWID,
                 unsigned int MeasRequestID,
                 unsigned int TimeOut,
                 TSMW_IQ_RESULT *pIQResult,
                 short* pReal, short* pImag,short* pScaling,
                 unsigned long* pOvfl,
                 unsigned int *pCalibrated,
                 unsigned int NoOfSamples,
                 unsigned int NoOfChannels,
                 int Reserved1, int Reserved2);
```

B.16 TSMWIQGetDataSingleIlv

Description:

This function unpacks the measurement result into interleave single precision floating point array format.

Delete measurement result

The function will delete the corresponding measurement result.

See also TSMWIQGetDataInt16.

Command syntax:

```
[ErrorCode,IQResultParam,
IQData,Scaling,Overflow,
            Calibrated] = TSMWIQGetDataSingle(TSMWID,
                                 MeasRequestID,TimeOut,
                                 NoOfSamples,NoOfChannels);
```

Parameter(s):

See TSMWIQGetDataInt16

Return value(s):

`ErrorCode`	See TSMWIQGetDataInt16.
`IQResultParam`	See TSMWIQGetDataInt16.
`IQData`	Complex I/Q data array.
`Scaling`	See TSMWIQGetDataInt16.
`Overflow`	See TSMWIQGetDataInt16.
`Calibrated`	See TSMWIQGetDataInt16.

Corresponding C++ command:

The parameters `Reserved1` and `Reserved2` have to be zero.

```
int ErrorCode = TSMWIQGetDataSingleIlv_c(
                 unsigned short TSMWID,
                 unsigned int MeasRequestID,
                 unsigned int TimeOut,
                 TSMW_IQ_RESULT_t *pIQResult,
                 float* pCplxData, short* pScaling,
                 unsigned long* pOvfl,
                 unsigned int *pCalibrated,
                 unsigned int NoOfSamples,
                 unsigned int NoOfChannels,
                 int Reserved1, int Reserved2);
```

B.17 TSMWIQGetDataDoubleIlv

Description:

This function unpacks the measurement result into interleave double precision floating point array format. The corresponding measurement result will be deleted.

Delete measurement result

The function will delete the corresponding measurement result.

See also TSMWIQGetDataInt16.

Command syntax:

```
[ErrorCode,IQResultParam,
IQData,Scaling,Overflow,
Calibrated] = TSMWIQGetDataDouble(TSMWID,MeasRequestID,
                             TimeOut,NoOfSamples,NoOfChannels);
```

Parameter(s):

See TSMWIQGetDataInt16.

Return value(s):

`ErrorCode`	See TSMWIQGetDataInt16.
`IQResultParam`	See TSMWIQGetDataInt16.
`IQData`	See TSMWIQGetDataSingleIlv.
`Scaling`	See TSMWIQGetDataInt16.
`Overflow`	See TSMWIQGetDataInt16.
`Calibrated`	See TSMWIQGetDataInt16.

Corresponding C++ command:

The parameters `Reserved1` and `Reserved2` have to be zero.

```
int ErrorCode = TSMWIQGetDataDoubleIlv_c(
           unsigned short TSMWID,unsigned int MeasRequestID,
           unsigned int TimeOut,TSMW_IQ_RESULT_t *pIQResult,
           double* pCplxData, short* pScaling,
           unsigned long* pOvfl, unsigned int *pCalibrated,
           unsigned int NoOfSamples,
           unsigned int NoOfChannels,
           int Reserved1, int Reserved2);
```

B.18 TSWMGetFIRParam

Description:

The function assists for custom filter design. It calculates the maximum number of FIR coefficients and an appropriate filter oversampling factor given a specific down sampling factor. The number of FIR filter tabs is given by `NoOfCoeffs/CoeffOver`.

Command syntax:

```
[NoOfCoeffs,OvsplFact] = TSWMGetFIRParam(Ndown, OvsplFact);
```

Parameter(s):

`Ndown`	Defines the fractional down sampling factor. Given the desired sampling rate fs, the down sampling factor is defined as fs/(395e6/18). I.e.: The desired sampling rate divided by the native R&S TSMW sampling rate.
`OvsplFact`	If given, the specified oversampling factor is used. If not, an appropriate oversampling factor is used. It has to be 1, 2, 4, 8, 16 or 32. This parameter is optional.

Return value(s):

`NoOfCoeffs`	Maximum number of FIR filters coefficients. The number of FIR filter tabs is given by NoOfCoeffs / OvsplFact.
`OvsplFact`	Selected oversampling factor.

Corresponding C++ command:

-/-

B.19 TSMWShutdown

Description:

The function shuts the specified R&S TSMW down.

Command syntax:

```
[ErrorCode] = TSMWShutdown(TSMWID);
```

Parameter(s):

`TSMWID`	Defines the TSMW to shut down.

Return value(s):

`ErrorCode`	0 if successful. Otherwise the error code is returned.

Corresponding C++ command:

```
int ErrorCode = TSMWShutdown_c(unsigned short TSMWID);
```

B.20 TSMWExtRefEnable

Description:

The function enables or disables usage of external reference clock.

Connect external reference clock before calling the function

If no external reference clock is connected and the function `TSMWExtRefEnable.m` is executed, measurements on RF 2 will not work.

Therefore the external reference has already connected before the function `TSMWExtRefEnable.m` is called.

Command syntax:

```
[ErrorCode] = TSMWExtRefEnable(TSMWID, Enable);
```

Parameter(s):

`TSMWID`	Defines the used TSMW.
`Enable`	1: Enable usage of external reference clock
	0: Disable

Return value(s):

`ErrorCode`	0 if successful. Otherwise the error code is returned.

Corresponding C++ command:

```
int ErrorCode = TSMWTGEnable_c(unsigned short TSMWID,
                                                  int Enable);
```

B.21 TSMWTGEnable

Description:

The function enables the tracking (test) generator output.

Command syntax:

```
[ErrorCode] = TSMWTGEnable(TSMWID, Enable);
```

Parameter(s):

`TSMWID`	Defines the used TSMW.
`Enable`	1: Enable test output 0: Disable test output

Return value(s):

`ErrorCode`	0 if successful. Otherwise the error code is returned.

Corresponding C++ command:

```
int ErrorCode = TSMWTGEnable_c(unsigned short TSMWID,
                                    unsigned int Frontends);
```

B.22 TSMWSetDAC

Description:

The function set the R&S TSMW references oscillator DAC (Digital-to-Analog Converter) value for fine adjustment of the internal reference oscillator.

GPS compatibility

This function does only work when GPS synchronization is disabled.

Command syntax:

```
[ErrorCode] = TSMWSetDAC(TSMWID,Value);
```

Parameter(s):

`TSMWID`	Specifies the TSMW which shall be used for streaming.
`Value`	Specifies DAC value. Value range: 0..1023. Reasonable values are in the range 200..800.

Return value(s):

`ErrorCode`	0 if successful. Otherwise the error code is returned.

Corresponding C++ command:

```
int ErrorCode = TSMWSetDAC_c(unsigned short TSMWID,
                                                   short Value);
```

B.23 TSMWGPSSync

Description:

The function enables or disables GPS synchronization. GPS synchronization is enabled by default.

Command syntax:

```
[ErrorCode] = TSMWGPSSync(TSMWID,Enable);
```

Parameter(s):

`TSMWID`	Specifies the TSMW ID.	
`Enable`	1	Enable GPS synchronization.
	0	Disable GPS synchronization.

Return value(s):

`ErrorCode`	0 if successful. Otherwise the error code is returned.

Corresponding C++ command:

```
int ErrorCode = TSMWGPSSync_c(unsigned short TSMWID,
                                                     int Enable);
```

B.24 TSMWGetIQTime

Description:

The function retrieves an estimate of the current I/Q time at R&S TSMW. The I/Q counter starts as soon as a connection to the R&S TSMW is established. The counter counts the I/Q-samples in the native sampling rate of `395/18 MS/s`. It has a valid bit width of `48 bit`.

The I/Q-time specifies the time that is used to start measurements.

Command syntax:

```
[ErrorCode, IQTime] = TSMWGetIQTime(TSMWID);
```

Parameter(s):

`TSMWID`	Defines the TSMW.

Return value(s):

`ErrorCode`	0 if successful. Otherwise the error code is returned.
`IQTime`	Specifies the current I/Q time.

Corresponding C++ command:

```
int ErrorCode = TSMWGetIQTime_c(unsigned short TSMWID,
                                unsigned __int64 *pIQTime);
```

B.25 TSMWTrigger

Description:

The function set and read R&S TSMW trigger I/O data.

Command syntax:

```
[ErrorCode,TriggerIn1,
       TriggerIn2]  = TSMWTrigger(TSMWID, TriggerLine, Out,
                             TriggerValue1, TriggerValue2);
```

Parameter(s):

`TSMWID`	Specifies the TSMW ID to use for measurement.
`TriggerLine`	Set or get data for different trigger lines:
	1: For trigger line 1.
	2: For trigger line 2.
	3: For trigger line 1 and 2.
`Out`	Set the corresponding trigger line:

	0: Set to 'input'. 1: Set to 'output'.
`TriggerValue1`	If trigger line 1 is used as output 0: set to low 1: set to high.
`TriggerValue2`	If trigger line 2 is used as output 0.set to low 1: set to high.

Return value(s):

`ErrorCode`	0 if successful. Otherwise the error code is returned.
`TriggerIn1`	Returns the current value of trigger line 1 (only if it is if used as input.) 0: low 1: high
`TriggerIn2`	Returns the current value of trigger line 1 (only if it is if used as input.) 0: low 1: high.

Corresponding C++ command:

```
int ErrorCode = TSMWTrigger_c(unsigned short TSMWID,
                 int TriggerLine,int Out,int *pTrig1Value,
                                       int *pTrig2Value);
```

B.26 TSMWGetVersion

Description:

The function returns the version number of the R&S TSMW IQ interface.

Command syntax:

```
[VersionNo] = TSMWGetVersion;
```

Parameter(s):

-/-

Return value(s):

`VersionNo`	Returns the version number in a 32-bit integer format. It represents the following 4-byte version code: MAJOR.MINOR.PATCH.QFE.

Corresponding C++ command:

```
int VersionNo = TSMWGetVersion_c();
```

B.27 TSMWGetVersionText

Description:

The function returns the version text of the R&S TSMW IQ interface.

Command syntax:

```
[VersionText] = TSMWGetVersionText;
```

Parameter(s):

-/-

Return value(s):

`VersionText`	Returns the version text of the R&S TSMW IQ Interface.

Corresponding C++ command:

```
char* VersionText = TSMWGetVersionText_c();
```

B.28 TSMWGPSEnable

Description:

The function enables or disables GPS data subscription from R&S TSMW. NMEA and UBX GPS data lines will be received from the R&S TSMW and are stored in a ring buffer of predefined size of 20 lines. If the buffer size is exceeded, the oldest received NMEA string or UBX message is overwritten. The NMEA and UBX buffers are independent.

Command syntax:

```
[ErrorCode] = TSMWGPSEnable(TSMWID,Enable);
```

Parameter(s):

`TSMWID`	Specifies the TSMW.
`Enable`	1: Enable GPS data subscription.
	0: Disable GPS data subscription.

Return value(s):

`ErrorCode`	0 if successful. Otherwise the error code is returned.

Corresponding C++ command:

```
int ErrorCode = TSMWGPSEnable_c(
                                 unsigned short TSMWID,
                                 int NoOfBufferedLines,
                                 int Enable);
```

B.29 TSMWGPSGetNMEALine

Description:

The function gets the oldest available NMEA text in buffer.

Configure NMEA buffer

The number of buffered NMEA text lines can be parameterized by TSMWGPSEnable_c. The MATLAB wrapper function TSMWGPSEnable uses a fixed buffer size of 20.

Command syntax:

```
[NMEAText] = TTSMWGPSGetNMEALine();
```

Parameter(s):

`-/-`

Return value(s):

`NMEAText`	Oldest available NMEA text in buffer.

Corresponding C++ command:

```
char NMEAText = TSMWGPSGetNMEALine_c();
```

B.30 TSMWGPSSendNMEACmd

Description:

The function sends NMEA command to the TSMW with the given TSMW ID.

Command syntax:

```
[ErrorCode] = TSMWGPSSendNMEACmd(TSMWID, NMEACmd);
```

Parameter(s):

`TSMWID`	Specifies the TSMW.
`NMEACmd`	NMEA text command.

Return value(s):

`ErrorCode`	0 if successful. Otherwise the error code is returned.

Corresponding C++ command:

```
int ErrorCode = TSMWGPSSendNMEACmd_c(
                                 int TSMWID, char* pNMEAString);
```

B.31 TSMWIQStream

Description:

The function starts a streaming measurement with given parameters. The measurement parameters are defined in the structure `MeasCtrl`.

Command syntax:

```
[ErrorCode] = TSMWIQStream(TSMWID,
                           MeasCtrl,FileName,CreateIfExists);
```

Parameter(s):

`TSMWID`	Specifies the TSMW to use for measurement.
`MeasData`	Measurement control structure. A template of the measurement control structure is provided by the MATLAB scripts “MeasCtrlTemplate”, “MeasCtrlTemplate_RF1” and “MeasCtrlTemplate_RF2”. If the field “ChannelCtrl1” is present in the MeasCtrl structure, a streaming measurement on RF-channel 1 is performed. If the field “ChannelCtrl2” is present, a streaming measurement on RF-channel 2 is performed. A streaming measurement on both channels is performed if both fields are present. For further information about the fields refer to chapter Structure: MeasCtrl on page 24 and provided MATLAB example scripts.
`FileName`	Specifies the file where the streaming data has to be saved. If no file name is given, online-processing of the streaming data is enabled. **Note:** Online-processing of streaming data includes unpacking the compressed stream data which needs additional processor performance and increases the amount of data. Highest streaming performance can be achieved when streaming to hard disk.
`CreateIf Exists`	Specifies whether an already existing streaming data file will be overwritten. 1: If a stream data file with the given file name already exists, it will be overwritten, if not a new file will be created. 0: If a stream data file with the given file name already exists, the function returns an error code.

Return value(s):

`ErrorCode`	0 if successful. Otherwise the error code is returned.

Corresponding C++ command:

```
int ErrorCode = TSMWIQStream_c(
                unsigned short TSMWID,
                TSMW_IQIF_MEAS_CTRL *pMEAS_CTRL,
                TSMW_IQIF_CH_CTRL *pCHANNEL_CTRL1,
                TSMW_IQIF_CH_CTRL *pCHANNEL_CTRL2,
                TSMW_IQIF_STREAM_CTRL_t *pStreamCtrl,
                char *pFileName, char *pDescription,
                unsigned int Flags);
```

B.32 TSMWIQStreamStatus

Description:

The function returns current streaming status.

Command syntax:

```
ErrorCode,StreamStatus] = TSMWIQStreamStatus(StreamID);
```

Parameter(s):

`StreamID`	Stream ID this file was associated with.

Return value(s):

`ErrorCode`	0 if successful. Otherwise the error code is returned.
`StreamStatus`	Streaming status structure. For further information about the fields refer to chapter Structure: StreamStatus on page 34 and provided MATLAB example scripts

Corresponding C++ command:

```
int ErrorCode = TSMWIQStopStreaming_c (
                 unsigned short TSMWID,
                 unsigned char StreamID,
                 TSMW_IQIF_STREAM_STATUS_t *pStreamStatus);
```

B.33 TSMWIQStopStreaming

Description:

The function stops a streaming measurement.

Command syntax:

```
[ErrorCode,StreamStatus] = TSMWIQStopStreaming(
                                          TSMWID, StreamID);
```

Parameter(s):

`TSMWID`	ID of R&S TSMW to use for streaming measurement.
`StreamID`	Stream ID that was specified with function TSMWIQStream on page 101.

Return value(s):

`ErrorCode`	0 if successful. Otherwise the error code is returned.
`StreamStatus`	Streaming status after streaming has stopped. For further information about the fields refer to chapter Structure: StreamStatus on page 34 and provided MATLAB example scripts.

Corresponding C++ command:

```
int ErrorCode = TSMWIQStopStreaming_c (
                  unsigned short TSMWID,
                  unsigned char StreamID,
                  TSMW_IQIF_STREAM_STATUS_t *pStreamStatus);
```

B.34 TSMWIQOpenStreamFile

Description:

The function opens a stream data file.

Command syntax:

```
[ErrorCode] = TSMWIQOpenStreamFile(FileName,StreamID);
```

Parameter(s):

`FileName`	Specifies the stream data file to open
`StreamID`	Stream ID this file was associated with.

Return value(s):

`ErrorCode`	0 if successful. Otherwise the error code is returned.

Corresponding C++ command:

```
int ErrorCode = TSMWIQOpenStreamFile_c(
                    char *pFileName, unsigned char StreamID,
                    TSMW_IQIF_STREAM_INFO_t *pStreamInfo,
                    TSMW_IQIF_MEAS_CTRL_t *pMeasCtrl,
                    TSMW_IQIF_CH_CTRL_t *pChannelCtrl1,
                    TSMW_IQIF_CH_CTRL_t *pChannelCtrl2,
                    TSMW_IQIF_FILTER_PARAM_t *pFilterSpec,
                    long *pCoeff,unsigned long NoOfCoeffs);
```

B.35 TSMWIQCloseStreamFile

Description:

The function closes the stream data file.

Command syntax:

```
[ErrorCode] = TSMWIQCloseStreamFile(StreamID);
```

Parameter(s):

`StreamID`	Stream ID this file was associated with.

Return value(s):

`ErrorCode`	0 if successful. Otherwise the error code is returned.

Corresponding C++ command:

```
int ErrorCode = TSMWIQStopStreaming_c (
                  unsigned short TSMWID,
                  unsigned char StreamID,
                  TSMW_IQIF_STREAM_STATUS_t *pStreamStatus);
```

B.36 TSMWIQGetStreamSingle

Description:

This function unpacks the streaming measurement result into single precision floating point array format. The function works for both "online" streaming and "offline" streaming.

"Online" streaming means processing stream data on the fly, hence no automatic writing to a stream data file. During "online" streaming the memory for the stream data is released automatically.

"Offline" streaming means that data is recorded on the disc in a stream data file. After the streaming has been finished, the stream data file can be opened and data can be read from the file using this function.

Command syntax:

```
[ErrorCode,
IQResultParam,
IData,QData,Scaling,Overflow,
Calibrated]     = TSMWIQGetDataStreamSingle(StreamID,Timeout
                             Offset,NoOfSamples,NoOfChannels);
```

Parameter(s):

`StreamID`	Stream ID that was specified with function TSMWIQStream on page 101.
`TimeOut`	Defines the time in milliseconds to wait for the measurement response result.
`Offset`	Sample offset where to get I/Q data. Only used for offline streaming (reading from a stream data file).
`NoOfSamples`	Defines the number of expected samples. This has to be equal to the number of samples specified in the measurement request (TSMWIQStream).
`NoOfChannels`	Specifies the number of expected channels. This has to be equal to the total number of channels the corresponding measurement request covers. I.e.: Measuring only on frontend 1 with only 1 sub-channel means NoOfChannels = 1. Measuring on frontend 1 and 2 with 1 sub-channel active on frontend 1 and 2 sub-channels on frontend 2 mean NoOfChannels = 3.

Return value(s):

See TSMWIQGetDataInt16.

Corresponding C++ command:

```
int ErrorCode = TSMWIQGetStreamSingle_c(
                    unsigned char StreamID,
                    unsigned int TimeOut,
                    TSMW_IQ_RESULT_t *pIQResult,
                    float* pReal, float* pImag,
                    short* pScaling, unsigned long* pOvfl,
                    unsigned int *pCalibrated,
                    unsigned __int64 Offset
                    unsigned int NoOfSamples,
                    unsigned int NoOfChannels);
```

B.37 TSMWIQGetStreamDouble

Description:

This function unpacks the streaming measurement result into double precision floating point array format. The function works for both "online" streaming and "offline" streaming. For more details about "online" and "offline" streaming see description of TSMWIQGetStreamSingle on p. 105.

Command syntax:

```
[ErrorCode,IQResultParam,
IQData,Scaling,Overflow
Calibrated]= TSMWIQGetStreamDouble(StreamID,Timeout,
                          Offset,NoOfSamples,NoOfChannels);
```

Parameter(s):

See TSMWIQGetStreamSingle.

Return value(s):

See TSMWIQGetStreamSingle.

Corresponding C++ command:

```
int ErrorCode = TSMWIQGetStreamDouble_c(
                               unsigned char StreamID,
                               unsigned int TimeOut,
                               TSMW_IQ_RESULT_t *pIQResult,
                               double* pReal,
                               double* pImag,
                               short* pScaling,
                               unsigned long* pOvfl,
                               unsigned int *pCalibrated,
                               unsigned __int64 Offset
                               unsigned int NoOfSamples,
                               unsigned int NoOfChannels);
```

B.38 TSMWIQGetStreamSingleIlv

Description:

This function unpacks the streaming measurement result into interleave single precision floating point array format.

The function works for both "online" streaming and "offline" streaming. For more details about "online" and "offline" streaming see description of TSMWIQGetStreamSingle on p. 105.

Command syntax:

```
[ErrorCode,IQResultParam,
IQData,Scaling,Overflow,
Calibrated]=TSMWIQGetDataStreamSingle(StreamID,TimeOut,
                                 Offset,NoOfSamples,NoOfChannels);
```

Parameter(s):

`StreamID`	Stream ID that was specified with function TSMWIQStream on page 101.
`TimeOut`	Max. time in ms to wait for the measurement data.
`Offset`	Sample offset where to get I/Q data. Only used for offline streaming (reading from a stream data file).
`NoOfSamples`	Defines the number of expected samples. This has to be equal to the number of samples specified in the measurement request (TSMWIQStream).
`NoOfChannels`	Specifies the number of expected channels. This has to be equal to the total number of channels the corresponding measurement request covers. I.e.: Measuring only on frontend 1 with only 1 sub-channel means NoOfChannels = 1. Measuring on frontend 1 and 2 with 1 sub-channel active on frontend 1 and 2 sub-channels on frontend 2 mean NoOfChannels = 3.

Return value(s):

`ErrorCode`	See TSMWIQGetDataInt16.
`IQResultParam`	See TSMWIQGetDataInt16.
`IQData`	Complex I/Q data array.
`Scaling`	See TSMWIQGetDataInt16.
`Overflow`	See TSMWIQGetDataInt16.
`Calibrated`	See TSMWIQGetDataInt16.

Corresponding C++ command:

```
int ErrorCode = TSMWIQGetStreamSingleIlv_c(
                              unsigned char StreamID,
                               unsigned int TimeOut,
                               TSMW_IQ_RESULT_t *pIQResult,
                               float* pCplxData,
                               short* pScaling,
                               unsigned long* pOvfl,
                               unsigned int *pCalibrated,
                               unsigned __int64 Offset,
                               unsigned int NoOfSamples,
                               unsigned int NoOfChannels);
```

B.39 TSMWIQGetStreamDoubleIlv

Description:

This function unpacks the streaming measurement result into interleave double precision floating point array format.

The function works for both "online" streaming and "offline" streaming. For more details about "online" and "offline" streaming see description of TSMWIQGetStreamSingle on page 105.

Command syntax:

```
[ErrorCode,IQResultParam,
IQData,Scaling,Overflow,
Calibrated]= TSMWIQGetStreamDouble(StreamID,Timeout,Offset,
                                NoOfSamples, NoOfChannels);
```

Parameter(s):

See TSMWIQGetStreamSingle.

Return value(s):

See TSMWIQGetStreamSingle.

Corresponding C++ command:

```
int ErrorCode = TSMWIQGetStreamDoubleIlv_c(
                    unsigned char StreamID,
                    unsigned int TimeOut,
                    TSMW_IQ_RESULT *pIQResult,
                    double* pCplxData, short* pScaling,
                    unsigned long* pOvfl,
                    unsigned int *pCalibrated,
                    unsigned __int64 Offset
                    unsigned int NoOfSamples,
                    unsigned int NoOfChannels);
```

B.40 MeasCtrlTemplate

Description:

The function creates a template for the `MeasCtrl` structure for double channel (MIMO) measurements with default values.

Command syntax:

```
[MeasCtrl] = MeasCtrlTemplate();
```

Parameter(s):

-/-

Return value(s):

`MeasCtrl`	Specifies a `MeasCtrl` structure with default values for double channel (MIMO) measurements. This means that the general measurement control sub-structure, the sub-structure for RF channel 1 as well as the sub-structure for RF channel 2 parameters will be set. For a detailed description of the structure refer to chapter chapter Structure: MeasCtrl on page 24.

Corresponding C++ command:

-/-

B.41 MeasCtrlTemplate_RF1

Description:

The function creates a template for the `MeasCtrl` structure for RF channel 1 measurements.

Command syntax:

```
[MeasCtrl] = MeasCtrlTemplate_RF1();
```

Parameter(s):

-/-

Return value(s):

`MeasCtrl`	Specifies a `MeasCtrl` structure with default values for RF channel 1 measurements. This means that the general measurement control sub-structure and the sub-structure for RF channel 1 parameters will be set. For a detailed description of the structure refer to chapter Structure: MeasCtrl on page 24.

Corresponding C++ command:

-/-

B.42 MeasCtrlTemplate_RF2

Description:

The function creates a template for the `MeasCtrl` structure for RF channel 2 measurements.

Command syntax:

```
[MeasCtrl] = MeasCtrlTemplate_RF2();
```

Parameter(s):

-/-

Return value(s):

`MeasCtrl`	Specifies a `MeasCtrl` structure with default values for RF channel 2 measurements. This means that the general measurement control sub-structure and the sub-structure for RF channel 2 parameters will be set. For a detailed description of the structure refer to chapter Structure: MeasCtrl on page 24.

Corresponding C++ command:

-/-

B.43 MeasCtrlTemplate_RF1Stream

Description:

The function creates a template for the `MeasCtrl` structure for RF channel 1 streaming measurements.

Resampling filter specification

In order to make use of this template, an appropriate resampling filter specification has to be transferred to the R&S TSMW before a streaming measurement is started. For filter design you might use the provided R&S TSMW Filter Design Tool.

Command syntax:

```
[MeasCtrl] = MeasCtrlTemplate_RF1Stream();
```

Parameter(s):

-/-

Return value(s):

`MeasCtrl`	Specifies a `MeasCtrl` structure with default values for RF channel 1streaming measurements. This means that the general measurement control sub-structure and the sub-structure for RF channel 1 parameters will be set. For a detailed description of the structure refer to chapter chapter Structure: MeasCtrl on page 24.

Corresponding C++ command:

-/-

B.44 MeasCtrlTemplate_RF2Stream

Description:

The function creates a template for the `MeasCtrl` structure for RF channel 2 streaming measurements.

Command syntax:

```
[MeasCtrl] = MeasCtrlTemplate_RF2Stream();
```

Parameter(s):

-/-

Return value(s):

`MeasCtrl`	Specifies a `MeasCtrl` structure with default values for RF channel 2streaming measurements. This means that the general measurement control sub-structure and the sub-structure for RF channel 2 parameters will be set. For a detailed description of the structure refer chapter Structure: MeasCtrl on page 24.

Corresponding C++ command:

-/-

Index