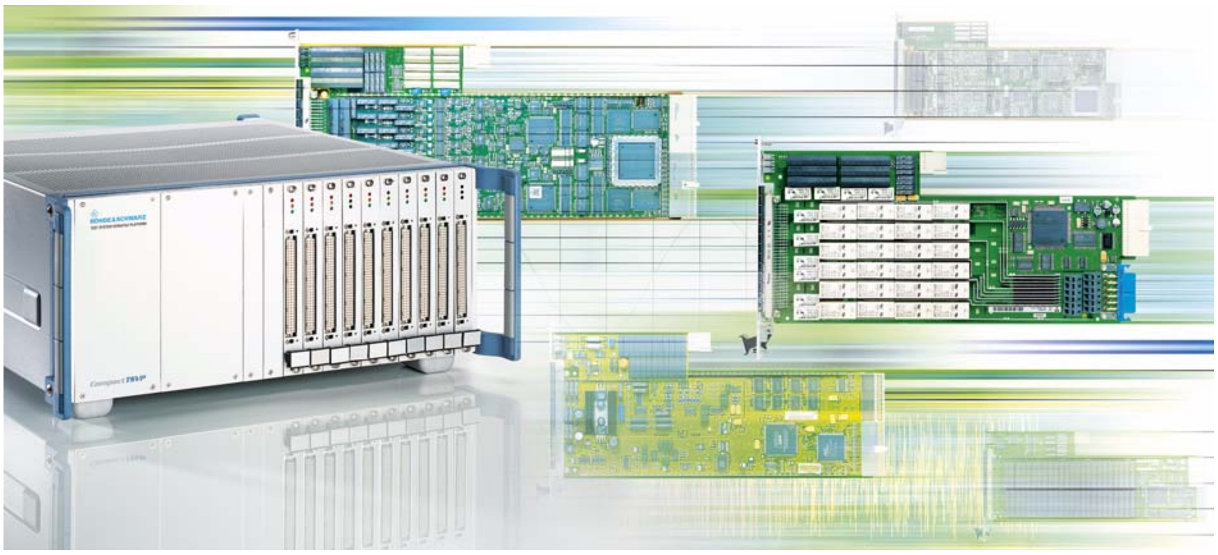
ROHDE & SCHWARZ

SOFTWARE DESCRIPTION



Generic Test Software Library R&S®GTSL

Software Description

for ROHDE & SCHWARZ Generic Test Software Library R&S GTSL

17th Issue / 08.09 / D 1143.6450.42

This Software Description is valid for the R&S GTSL-Software version **02.90** and higher.

ROHDE & SCHWARZ GmbH & Co. KG		
Corporate Headquarters	Telephone:	...49 (0)89/4129-13774
Mühldorfstr. 15	Fax:	...49 (0)89/4129-13777
D-81671 München		

Printed in the Federal Republic of Germany. Errors excepted, subject to technical change without notice.

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)

I ○	⏻	⎓	∼	≂	⧈
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

Address List

Headquarters, Plants and Subsidiaries

Headquarters

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

Plants

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

Subsidiaries

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

Locations Worldwide

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

- Sales Locations
- Service Locations
- National Websites

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 8:00 AM – 8:00 PM	(except US public holidays) Eastern Standard Time (EST)
	Tel. from USA From outside USA Fax	888-test-rsa (888-837-8772) (opt 2) +1 410 910 7800 (opt 2) +1 410 910 7801
	E-mail	CustomerSupport@rohde-schwarz.com
East Asia	Monday to Friday 8:30 AM – 6:00 PM	(except Singaporean public holidays) Singapore Time (SGT)
	Tel. Fax	+65 6 513 0488 +65 6 846 1090
	E-mail	CustomerSupport@rohde-schwarz.com
Rest of the World	Monday to Friday 08:00 – 17:00	(except German public holidays) Central European Time (CET)
	Tel. from Europe From outside Europe Fax	+49 (0) 180 512 42 42* +49 89 4129 13776 +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

Contents

17th Issue 08.09

Figures

Tables

1 General

The Generic Test Software Library R&S GTSL contains ready-to-use test cases for all important measurements and all common mobile phone standards. These include tests for all function groups of the mobile phone, such as audio and acoustic tests, HF test, calibration and signaling test.

The individual test cases are available as DLLs and can be individually adapted by a menu-driven process. An ASCII file contains the relevant configuration data which can be assigned to certain test sequences. So measurement parameters can be changed and adjusted easy and quickly with a standard editor.

TestStand software from National Instruments is used to control the test sequence. This software combines the individual test sequences to form an executable test program. It also adds all other functions important to the production operation, such as user administration, execution of multiple test sequences in multi-threading or parallel operation, collection and storage of relevant measurement results and report generation.

TestStand has an intuitive user interface and functions for executing and debugging test sequences. The integrated sequence editor allows you to create a test sequence easily by stringing together individual tests, and you can modify it at any time. The measurement data produced during the test sequence are collected and can be used for automatic generation of reports, or simply stored in a database for later evaluation.

The individual test cases of the Generic Test Software Library R&S GTSL can also be combined by other test sequence control systems (e.g. via a C program) into an executable test program.

NOTE:

Knowlege of the Windows 2000/XP operating system is needed to operate and work with the Generic Test Software Library R&S GTSL.
To use TestStand as the test sequence control system, you need to be familiar with the programs

- **LabWindows / CVI Run-Time Operator Interface for Test-Stand by National Instruments**
- **TestStand Sequence Editor by National Instruments.**

Knowledge of C-programming is needed to create your own test libraries.

2 Software installation

2.1 General

NOTE:

To install the <u>G</u>eneric <u>T</u>est <u>S</u>oftware <u>L</u>ibrary R&S GTSL under Windows 2000/XP, the user must be logged in as administrator or as a user with administrator rights.

For additional information on the deinstallation of previous versions of the Generic Test Software Library R&S GTSL or concerning installation, consult the README.TXT file on the installation CD.

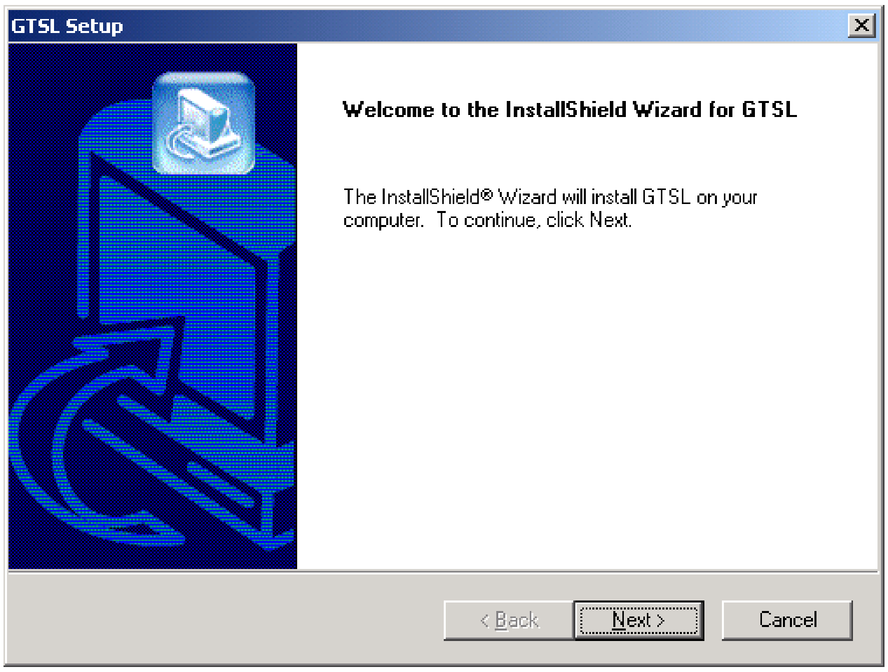
2.2 Installation

The Generic Test Software Library R&S GTSL is installed on the computer of the TSVP Test System Versatile Platform via an installation routine. Start the installation routine as follows:

1. Insert the CD containing the Generic Test Software Library R&S GTSL.
2. Start the installation routine by selecting **Start** -> **Run** and typing „X:\Setup“.

   X:\ is the drive letter of the CD-ROM drive.
3. Then follow the on-screen installation instructions.

A Welcome screen with installation wizard.

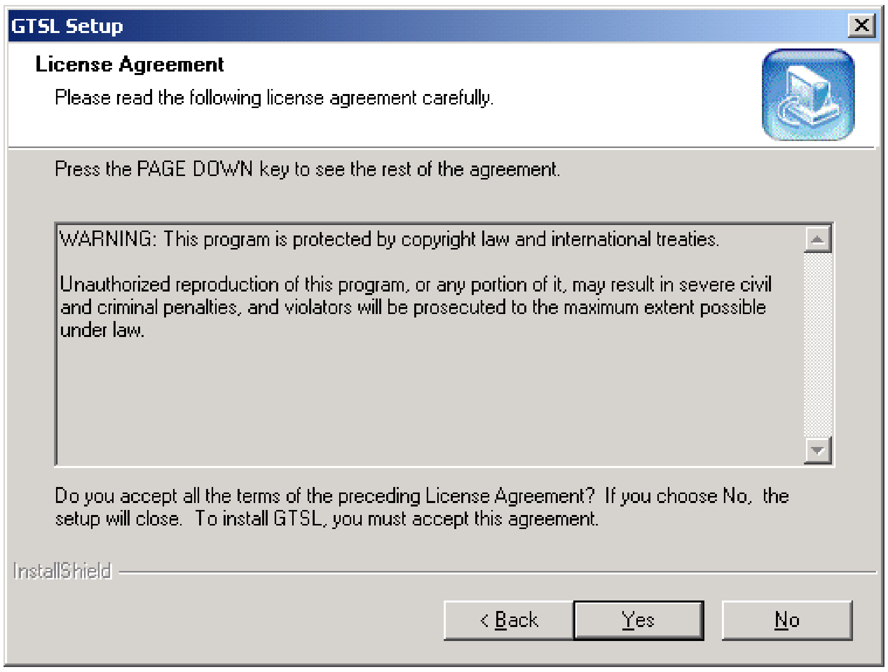

GTSL Setup

Welcome to the InstallShield Wizard for GTSL

The InstallShield® Wizard will install GTSL on your computer. To continue, click Next.

Figure 2-1 Setup Welcome Screen

B Accept the License Agreement.


GTSL Setup

License Agreement

Please read the following license agreement carefully.

Press the PAGE DOWN key to see the rest of the agreement.

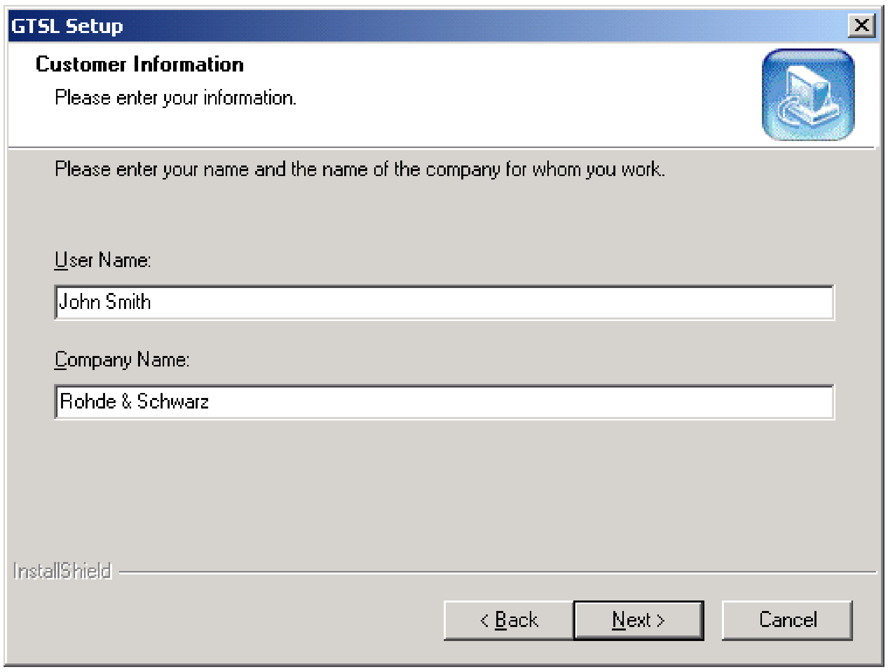
WARNING: This program is protected by copyright law and international treaties.

Unauthorized reproduction of this program, or any portion of it, may result in severe civil and criminal penalties, and violators will be prosecuted to the maximum extent possible under law.

Do you accept all the terms of the preceding License Agreement? If you choose No, the setup will close. To install GTSL, you must accept this agreement.

Figure 2-2 Setup License Agreement

C Enter a user name and company name.


GTSL Setup

Customer Information

Please enter your information.

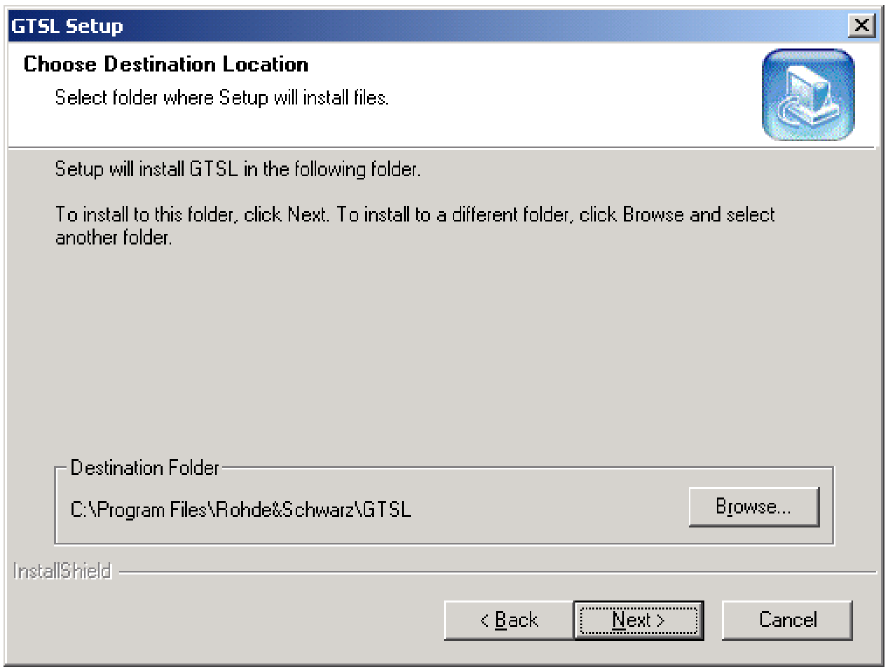
Please enter your name and the name of the company for whom you work.

User Name:
John Smith

Company Name:
Rohde & Schwarz

Figure 2-3 Setup User Information

D Select the directory where the R&S GTSL is to be installed.


GTSL Setup

Choose Destination Location

Select folder where Setup will install files.

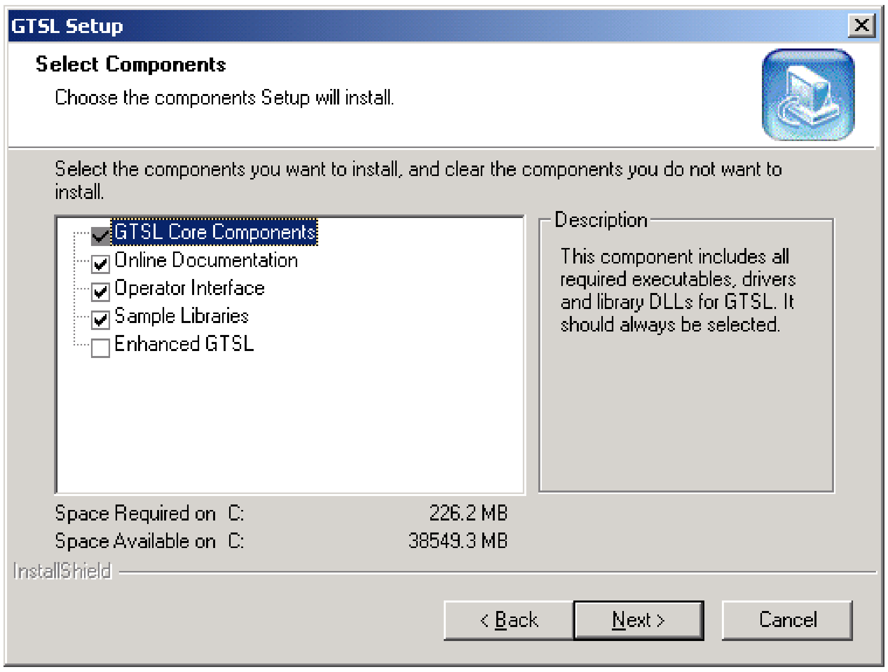
Setup will install GTSL in the following folder.

To install to this folder, click Next. To install to a different folder, click Browse and select another folder.

Destination Folder
C:\Program Files\Rohde&Schwarz\GTSL

Figure 2-4 Setup Choose Destination Location

E Select the program components to be install.


GTSL Setup

Select Components

Choose the components Setup will install.

Select the components you want to install, and clear the components you do not want to install.

- [✓] GTSL Core Components
- [✓] Online Documentation
- [✓] Operator Interface
- [✓] Sample Libraries
- [ ] Enhanced GTSL

Description

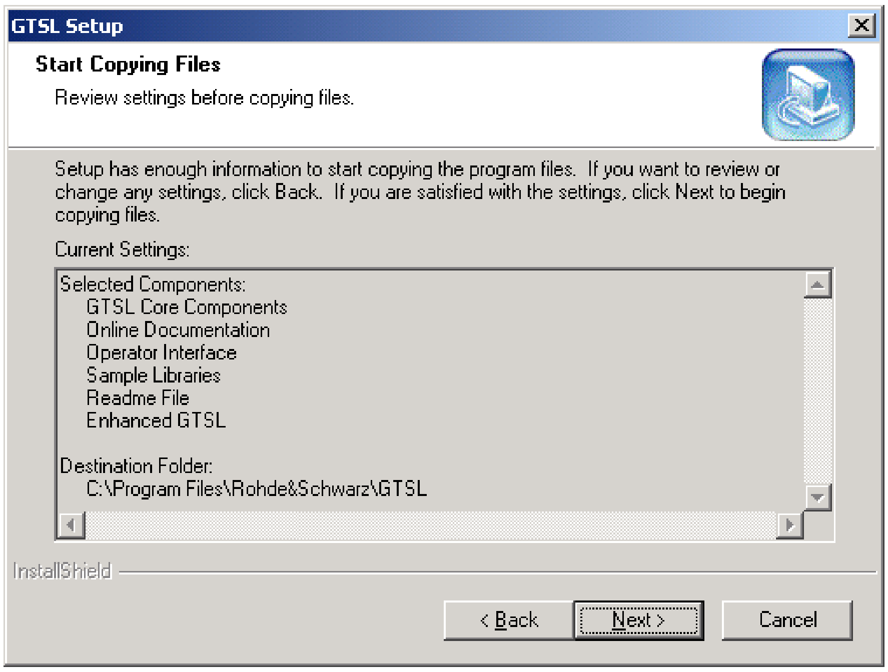
This component includes all required executables, drivers and library DLLs for GTSL. It should always be selected.

Space Required on C:	226.2 MB
Space Available on C:	38549.3 MB

InstallShield

Figure 2-5 Setup Select Program Components

F Display of the current setup settings.


GTSL Setup

Start Copying Files

Review settings before copying files.

Setup has enough information to start copying the program files. If you want to review or change any settings, click Back. If you are satisfied with the settings, click Next to begin copying files.

Current Settings:

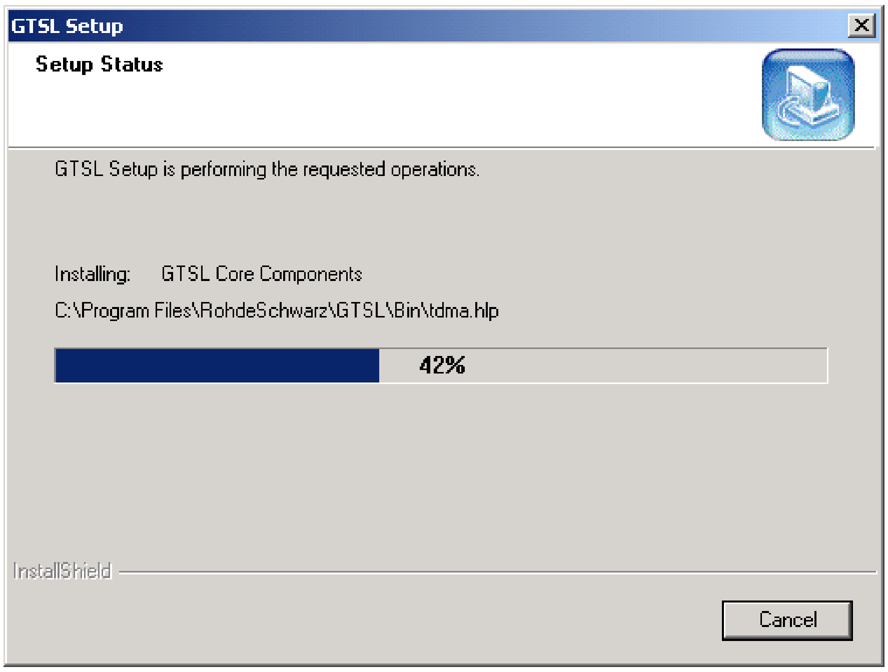
```
Selected Components:
    GTSL Core Components
    Online Documentation
    Operator Interface
    Sample Libraries
    Readme File
    Enhanced GTSL

Destination Folder:
    C:\Program Files\Rohde&Schwarz\GTSL
```

InstallShield

Figure 2-6 Setup Settings

17th Issue 08.09

G Display of the Setup Status


GTSL Setup

Setup Status

GTSL Setup is performing the requested operations.

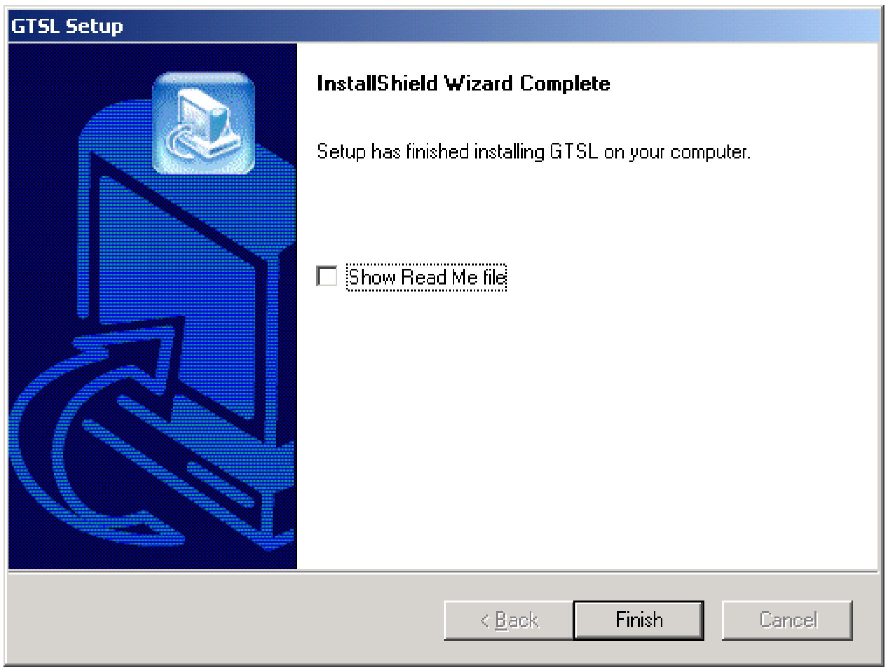
Installing: GTSL Core Components

C:\Program Files\RohdeSchwarz\GTSL\Bin\tdma.hlp

42%

Figure 2-7 Setup Status

H Close the installation routine.


GTSL Setup

InstallShield Wizard Complete

Setup has finished installing GTSL on your computer.

☐ Show Read Me file

Figure 2-8 Setup Complete

If you also need to install Acrobat Reader, check the relevant box. You need Acrobat Reader to display the documentation installed as PDF files.

During the installation process, the following programs are copied to the predefined file structure:

- Test libraries with help files
- Configuration files and calibration files
- Examples for the creation of test libraries
- Documentation
- Examples of test sequences

2.3 File structure

The test libraries and test sequences supplied by ROHDE & SCHWARZ are stored in fixed directories at the time of installation. The files from the corresponding directories are addressed from the Test-Stand Sequence Editor. ROHDE & SCHWARZ specify the following structure:

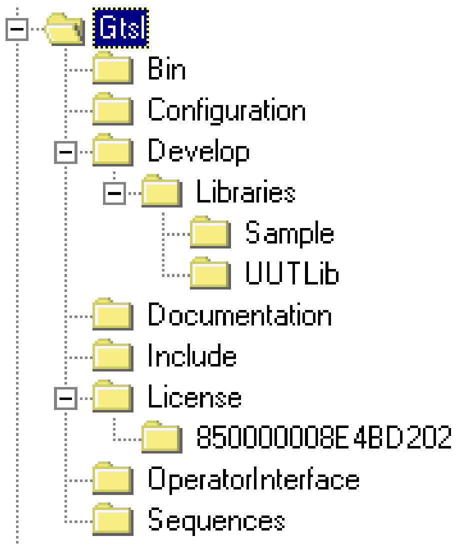


Figure 2-9 File structure

NOTE:

The file structure and the directory names below the R&S GTSL directory must be maintained.

Directory	Contents
GTSL	Generic Test Software Library. The root directory for the R&S GTSL software can have any name.
Bin	Contains the test libraries (DLL, LIB) and the help files belonging to the test libraries (HLP).
Configuration	Contains the two configuration files PHYSICAL.INI and APPLICATION.INI and the calibration files.

Table 2-1 File structure

Directory	Contents

Develop
  Libraries
    Sample
    UUTLib
	The directory `..\Sample` contains a generally valid example for the creation of a test library. The directory `..\UUTLib` contains an extended example for the creation of a test library for customer-specific communication with the mobile phone being tested. The directories contain the C source code of the example libraries.

Documentation	Contains the various items of documentation in the form of PDF files.
Include	Contains the h-files (include files) needed for the development of new test libraries.
License Serial Number	Contains, for each installed iButton, a subdirectory with the corresponding serial number. The test library license key files belonging to the serial number are stored in the directory.
OperatorInterface	Contains the run time module for the operator interface of TestStand. A TestStand run time licence is required.
Sequences	Contains the test sequence examples created by ROHDE & SCHWARZ.

Table 2-1 File structure

3 Functional description

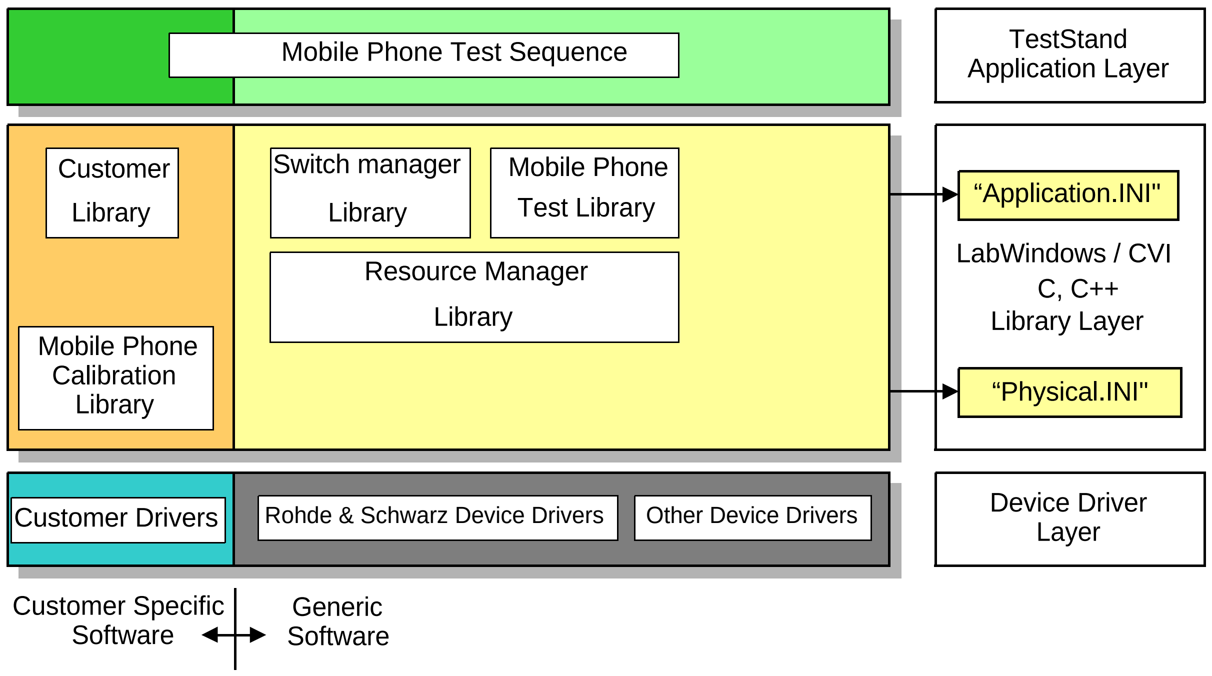

Mobile Phone Test Sequence
TestStand Application Layer
Customer Library
Switch manager Library
Mobile Phone Test Library
Resource Manager Library
Mobile Phone Calibration Library
"Application.INI"
LabWindows / CVI C, C++ Library Layer
"Physical.INI"
Customer Drivers
Rohde & Schwarz Device Drivers
Other Device Drivers
Device Driver Layer
Customer Specific Software
Generic Software

Figure 3-1 R&S GTSL layer model (example for testing of mobile phones)

In terms of its structure, the Generic Test Software Library R&S GTSL developed by ROHDE & SCHWARZ is divided into different supply components and software layers.

A distinction is made between the software components supplied by ROHDE & SCHWARZ and the components which must be supplied or adapted by the customer. The software components to be provided by the customer may for example include

- device drivers,
- calibration data,
- test libraries and
- test sequences

specific to the customer and to the unit under test.

The software used in the R&S GTSL is divided into three different layers.

The lowest level of the R&S GTSL accommodates the device drivers needed for the hardware used (Device Driver Layer). These include the device drivers for

- the hardware developed and used by ROHDE & SCHWARZ.
- the standard hardware used.

The middle level of the R&S GTSL accommodates the different test libraries (Library Layer). These test libraries provide the functions needed to execute test sequences. At this level, further information concerning the two files PHYSICAL.INI and APPLICATION.INI is transferred to the Resource Manager Library. The different device drivers of the lowest level are called from this level.

The highest level accommodates the test sequences for the execution of the individual test functions (Application Layer). The test sequences call functions from the libraries in the middle level.

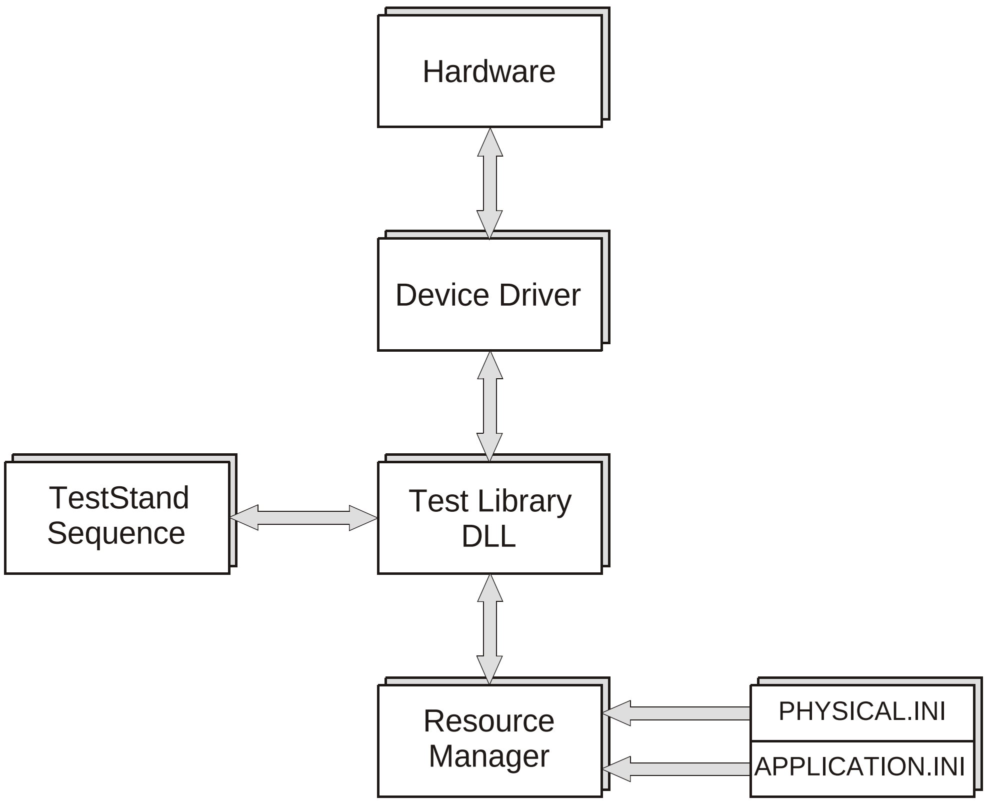


Figure 3-2 Software structure

The test libraries form the core of the software and of the test sequences. The test functions stored in the test libraries (Dynamic Link Library DLL) are combined within TestStand into executable test sequences.

The individual test functions access the test system hardware via the device drivers.

The hardware initialized in the Generic Test Software Library is managed by the Resource Manager. The Resource Manager is likewise a

DLL file.

Thanks to the hardware management of the Resource Manager, the created test sequences are independent of the current hardware configuration, so test sequences do not need to be modified if the hardware or hardware settings are changed. All information needed to run the test sequences is sent to the Resource Manager via two configuration files (INI files):

- PHYSICAL.INI
- APPLICATION.INI

Only these files need to be modified if the hardware or hardware settings are changed. They can be edited with any text editor.

The Resource Manager also manages the hardware during the parallel execution of test sequences. The Resource Manager prevents conflicts in accessing different test functions or test sequences on the same hardware.

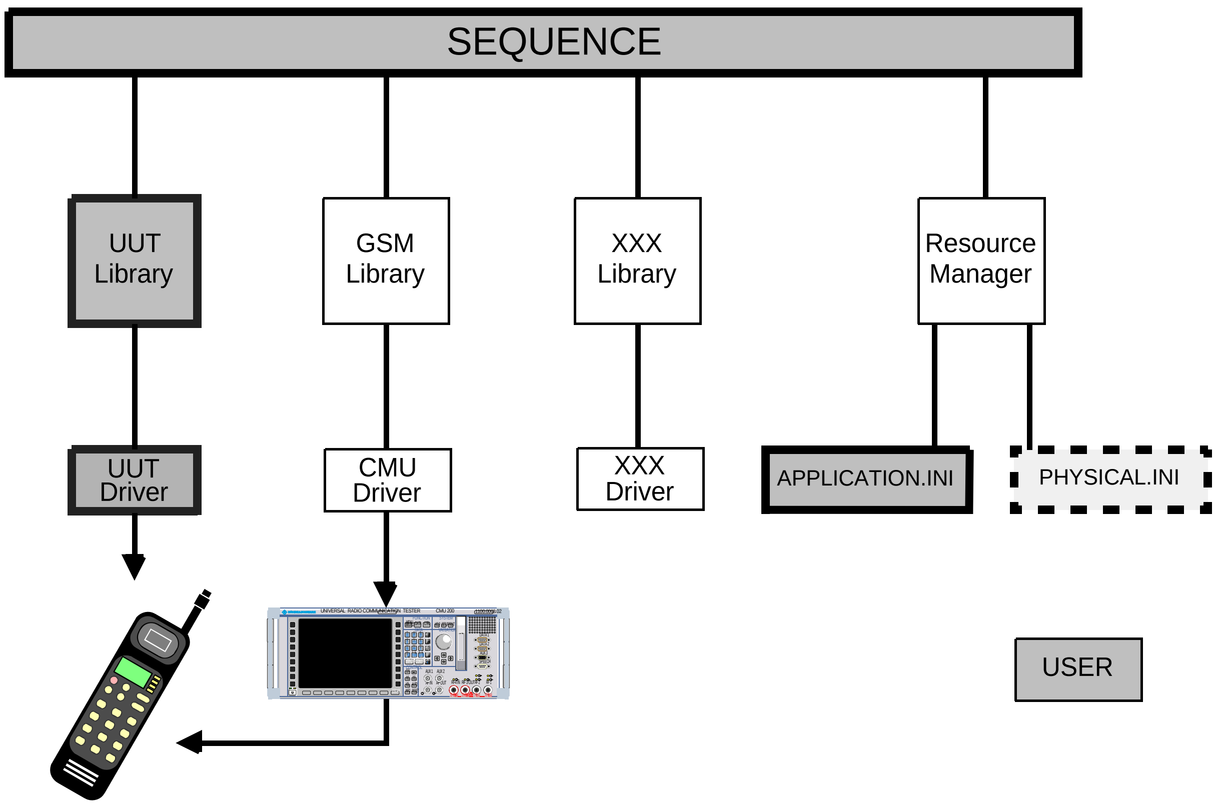


Figure 3-3 User components of R&S GTSL

When using the Generic Test Software Library R&S GTSL, the user only has to make changes to certain components of the software. In all cases, the user creates the test sequences for the test applications at the highest level (Application Layer).

At the middle level (Library Layer), the user utilises the functions of the test library. The UUT test library

- for direct communication with the mobile phone being tested
- for addressing special functions of the mobile phone being tested

must be adapted by the user to the type that is being tested. These adaptations are necessary because the required UUT device drivers and function calls are different for every type of mobile phone (e.g. chipsets).

The user has to adapt the configuration file APPLICATION.INI to the relevant test application. The user only has to adapt the configuration file PHYSICAL.INI in the event of a change to the hardware configuration.

☞ **NOTE:**

Since an APPLICATION.INI configuration file normally exists for every test application performed on the system, the file name can be matched to the test application in question, e.g. APP_XXX.INI. The Resource Manager is told during setup which configuration file is to be used.

On the other hand, only one PHYSICAL.INI configuration file is ever available on the system.

For ease of comprehension, the file names APPLICATION.INI and PHYSICAL.INI are used for the configuration files in the manual.

3.1 Operation of a test sequence

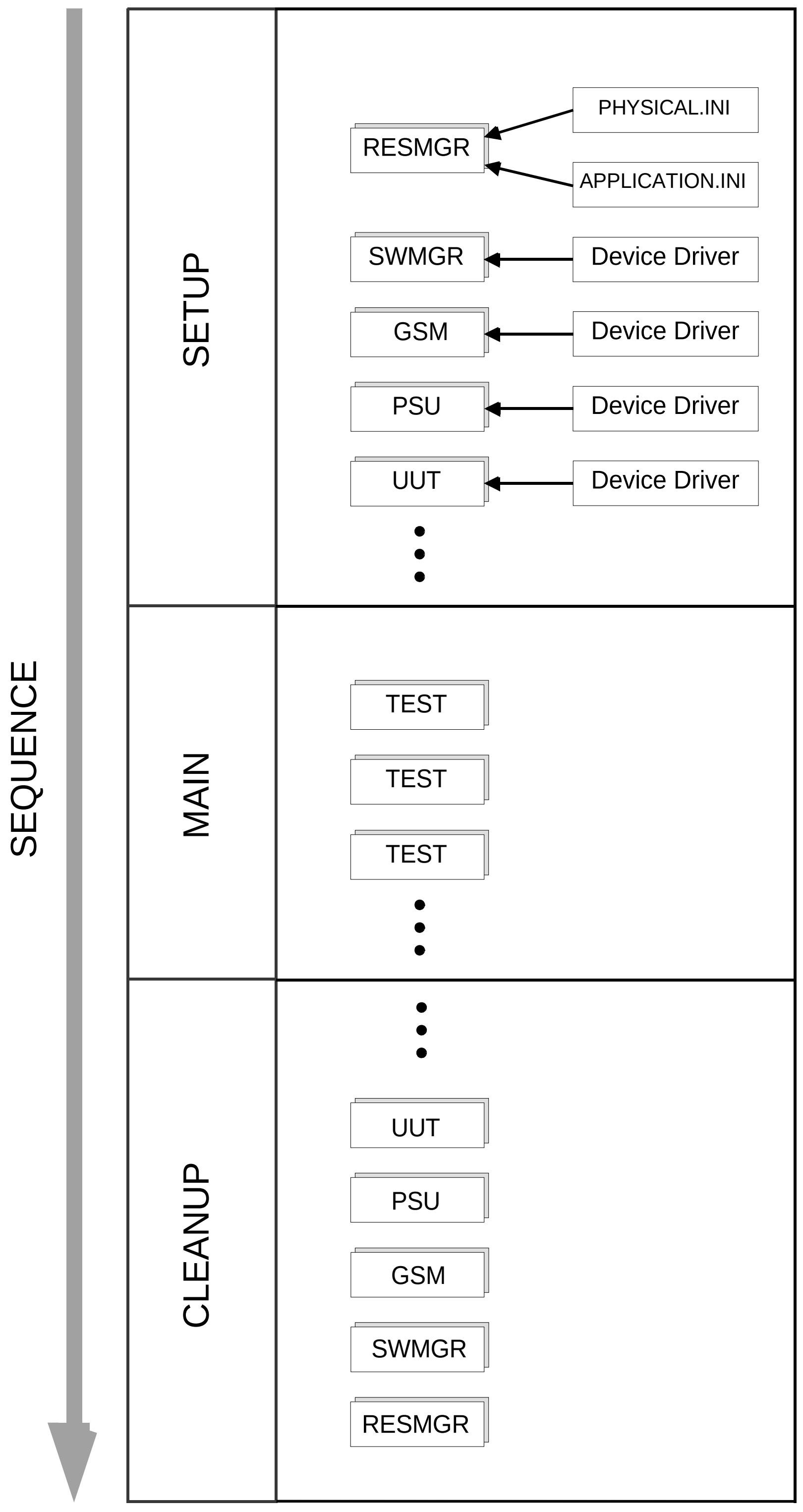


Figure 3-4 Test sequence operation

The operation of every created and executed test sequence is divided into three stages:

1. SETUP

   First of all, the SETUP function of the Resource Manager (RESMGR) is called. During this function call, information from the two configuration files PHYSICAL.INI and APPLICATION.INI is loaded. Then, the SETUP functions of the individual libraries needed to perform the test steps are called (SWMGR, GSM, PSU, UUT etc.). The necessary hardware and software components are requested, the relevant device drivers are initialized and the devices are placed in a defined state.

2. MAIN

   The individual test steps are performed.

3. CLEANUP

   The CLEANUP functions of the Resource Manager and of the used libraries are called. The system resources reserved during the setup functions and the reserved hardware are freed again. A CLEANUP call is needed for every SETUP call.

   The different CLEANUP function calls are executed even if the operation of the test steps is interrupted. This ensures that the used system resources are always freed again, and the used hardware is always returned to a defined basic state.

The division of the execution operation of a test sequence into these three subareas takes place within TestStand or in the test sequence control system that is used.

4 R&S GTSL license management

During the installation of the Generic Test Software Library R&S GTSL, all available test libraries are copied to the system. You need a License Key File in order to access the functions from the test libraries. Refer to chapter 7 for the license key required for each test library.

Without a valid License Key File, the functions of the test library will only work in „demo mode“. Access to the hardware is only simulated.

Each license is bound to a system serial number. The enabled test libraries can only be run on the system with this serial number. The hardware system used is identified via the system module PSYS (R&S CompactTSVP, R&S PowerTSVP) or via an iButton (TSVP, desktop PC, notebook). The iButton is located on a Hardlock-Adapter, which is plugged into the serial interface (usually COM1) of the system controller of the Test System Versatile Platform TSVP or of the connected control computer. The system module or iButton has a unique serial number whereby the hardware system can be unambiguously identified.

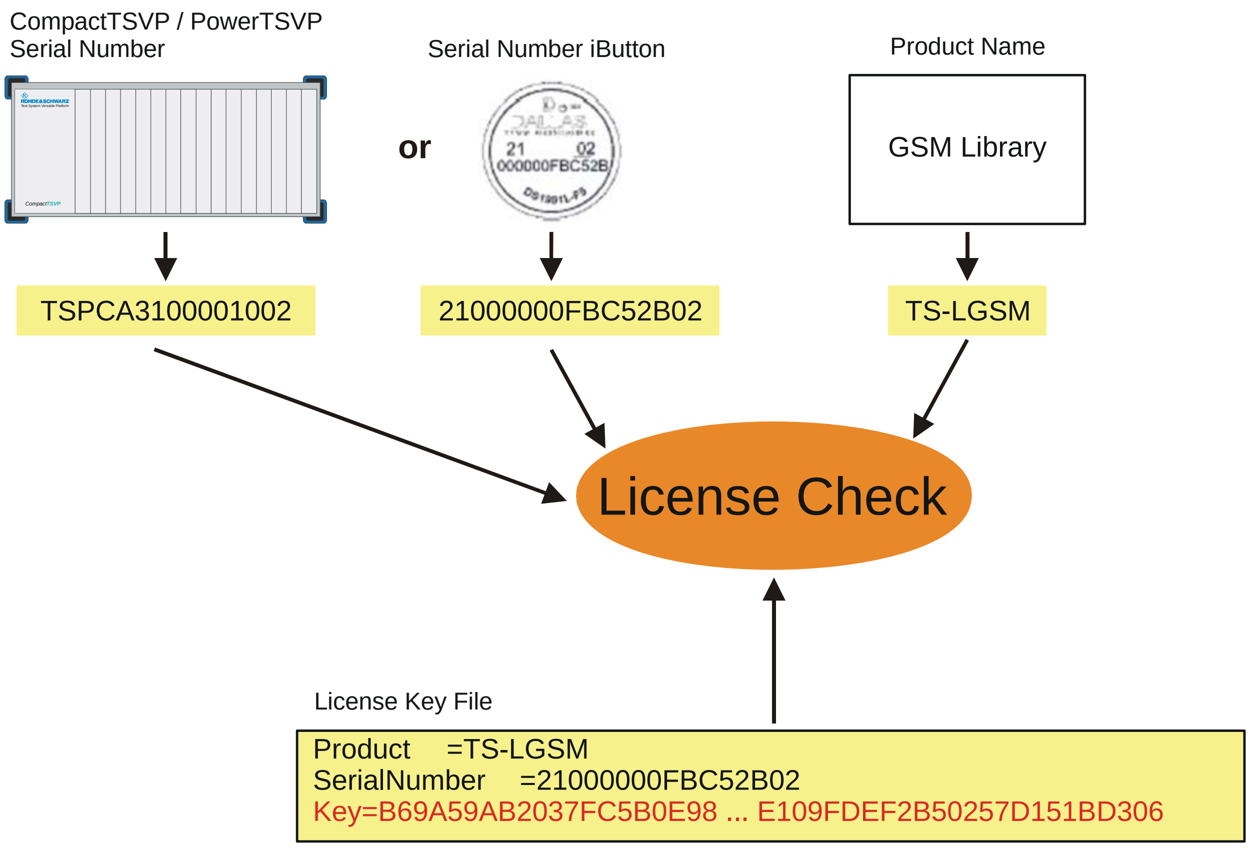


Figure 4-1 License checking

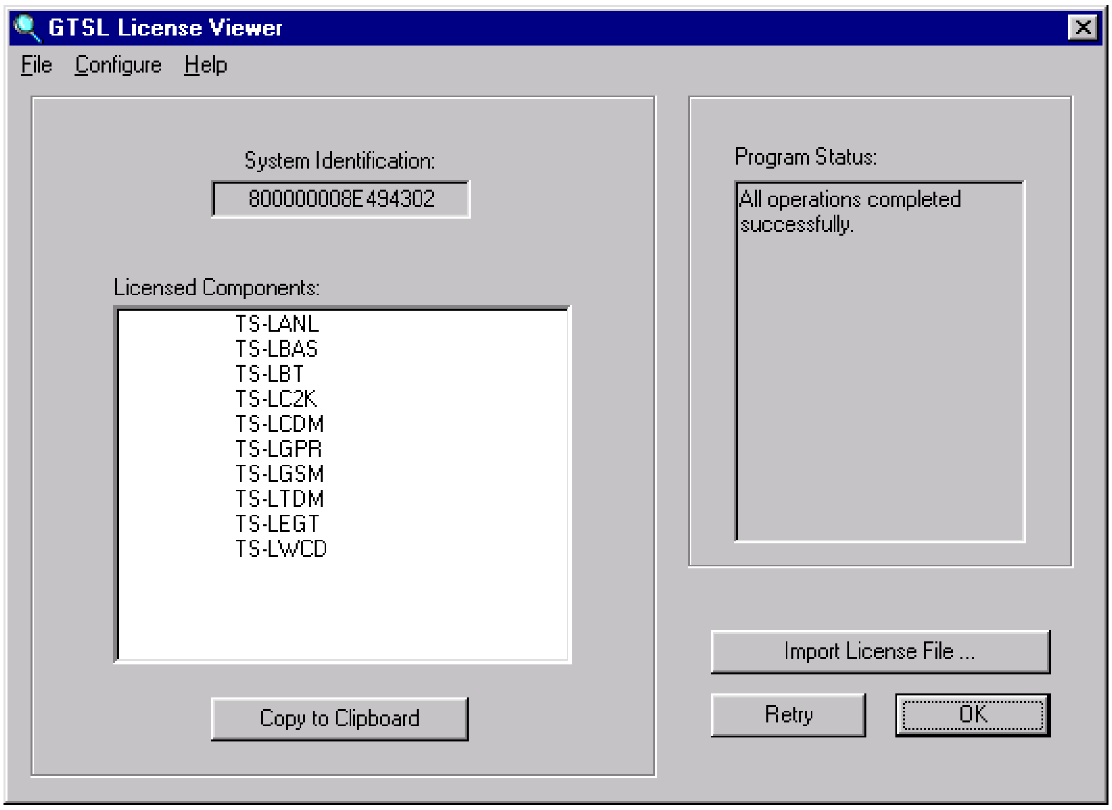
During license checking, the serial number and the name of the called test library are compared with the License Key from the License Key File. The test library in question will only be enabled if these coincide. The serial number and the name of the test library are encoded in the License Key.

Example of a License Key File:

```
[Header]
FileVer=1.0
[Project]
Info=GTSL
[Modul]
Product=TS-LBAS
SerialNumber=850000008E4BD202
Key=C146648E1DEF9AD78663728A5D8E8D25885F457367D7F7C359F2C63BDB926 ...
```

The serial number is queried and a new License Key File is installed via the R&S GTSL License Viewer. To open the R&S GTSL License Viewer, select

Start -> **Programs** -> **GTSL** -> **License Viewer**



800000008E494302

TS-LANL
TS-LBAS
TS-LBT
TS-LC2K
TS-LCDM
TS-LGPR
TS-LGSM
TS-LTDM
TS-LEGT
TS-LWCD

All operations completed successfully.

Figure 4-2 R&S GTSL License Viewer

System Identification	The serial number of the system (iButton or PSYS)
Program Status	Displays the current status of the R&S GTSL License Viewer.
Licensed Components	Displays the test libraries for which a License Key File has been imported.
Copy to Clipboard	Copies the displayed serial number to the clipboard.
Import License File ...	Imports a new License Key File into the R&S GTSL License Viewer. The path and the file name of the License Key File to be imported can be selected via a file browser dialog.
Retry	Reads the serial number into the R&S GTSL License Viewer.
OK	Closes the R&S GTSL License Viewer.
<File> ... <Exit>	Closes the R&S GTSL License Viewer.
<Configure> ... <System Identification>	Opens the configuration dialog (see Figure 4-3).
<Help> ... <About License Viewer>	Shows version and copyright information about the R&S GTSL License Viewer.

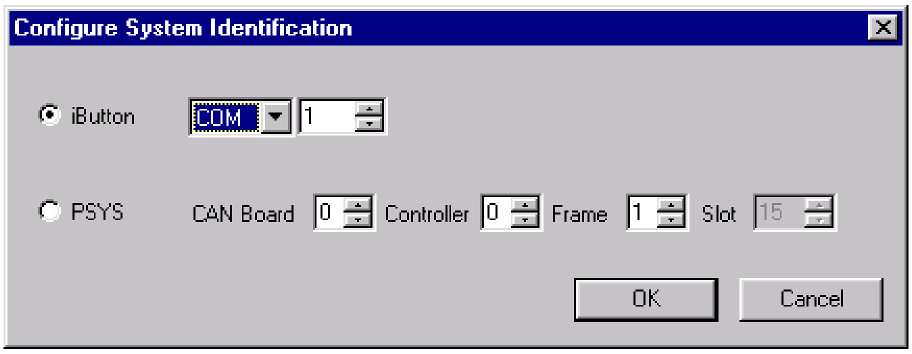


Figure 4-3 Configuration Dialog

iButton — Select the port type and port number where the iButton adapter is connected. Default = COM 1.
Note that LPT requires a special iButton adapter which is not part of the R&S GTSL shipment.

PSYS — "CAN Board" and "Controller" define the board and interface Id of the CAN interface controlling the R&S TS-PSYS module. Default = 0
"Frame" defines the R&S CompactTSVP or R&S PowerTSVP frame number where the R&S TS-PSYS module is located. Default = 1
"Slot": The only valid slot number for the R&S TS-PSYS module is 15.

OK — The settings made in the Configuration Dialog are accepted with the **OK** button.

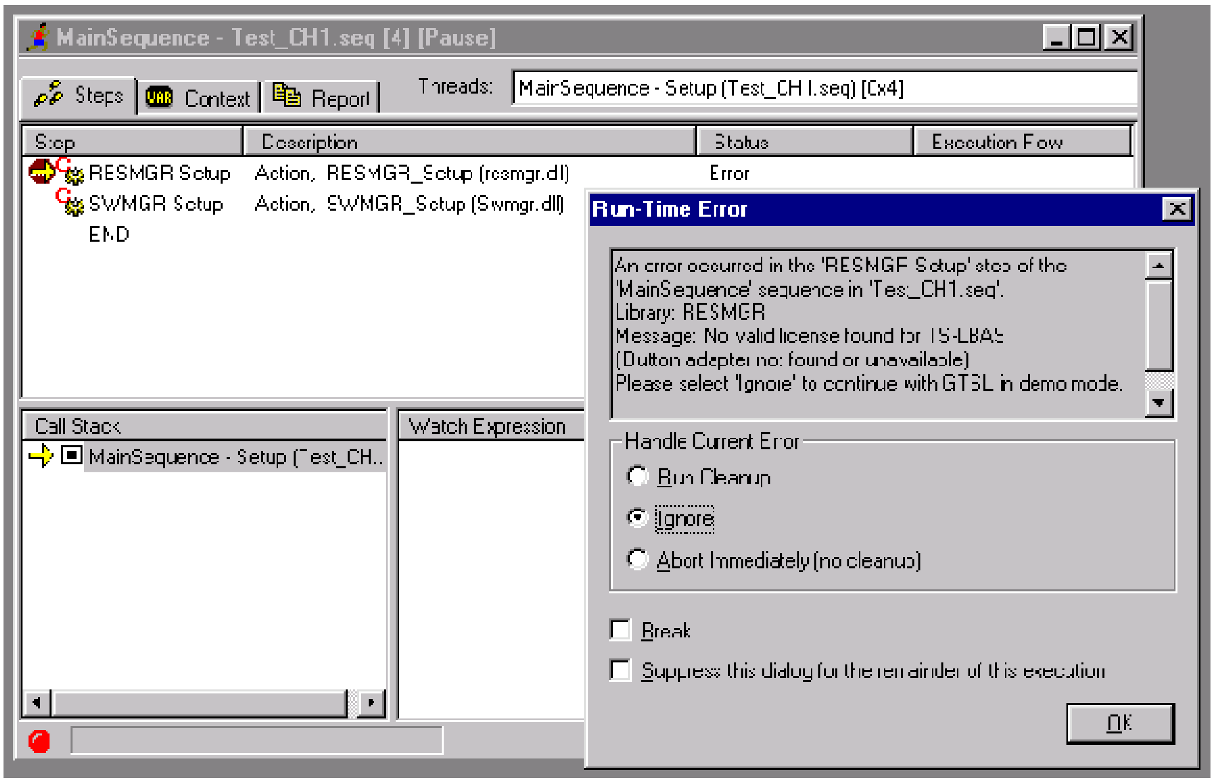
Cancel — The settings made in the Configuration Dialog are rejected with the **Cancel** button.

For each installed iButton or PSYS, a subdirectory with the relevant serial number is created in the `..\GTSL\License` directory. The License Key Files installed via the R&S GTSL License Viewer are stored in the relevant subdirectory.

If further test libraries are enabled, the serial number must be sent to ROHDE & SCHWARZ. To avoid writing errors, the serial number can be copied from the R&S GTSL License Viewer via the clipboard.

The License Key File newly created by ROHDE & SCHWARZ must be installed via the R&S GTSL License Viewer. After that, unrestricted use of the test libraries is possible.

If, during a test sequence, test functions are used or called from a non-enabled test library, a warning code will be displayed upon execution in TestStand. If non-enabled functions are called during Resource Manager Setup, an error message will be displayed:


MainSequence - Test_CH1.seq [4] [Pause]

Threads: MainSequence - Setup (Test_CH1.seq) [Cx4]

Step	Description	Status	Execution Flow
RESMGR Setup	Action, RESMGR_Setup (resmgr.dll)	Error	
SWMGR Setup	Action, SWMGR_Setup (Swmgr.dll)		
END			

Call Stack: MainSequence - Setup (Test_CH..

Run-Time Error

An error occurred in the 'RESMGR Setup' step of the 'MainSequence' sequence in 'Test_CH1.seq'.
Library: RESMGR
Message: No valid license found for TS-LBAS
(Dutton adapter not found or unavailable)
Please select 'Ignore' to continue with GTSL in demo mode.

Handle Current Error
- Run Cleanup
- [✓ Ignore]
- Abort Immediately (no cleanup)

Figure 4-4 Error message from Resource Manager

If the error message is acknowledged by selecting „Ignore“, the test sequence will run in „demo mode“.

5 Configuration files

The required configuration files are stored by default in the `..\GT-SL\Configuration` directory.

5.1 Syntax

The syntax of the Physical Layer Configuration File (PHYSICAL.INI) and of the Application Layer Configuration File (APPLICATION.INI) is identical. The only difference between the two files is in terms of how they are used (see chapter 5.2 and 5.3).

Both files use the standard INI file format.

Example of standard INI file format:

```
[section]

key = value
...
```

A section begins with the section name written inside closed brackets ([ ]). The following lines contain pairs of keywords and values. The keywords and the assigned values are separated by an equals sign („=“).

In the section names and keywords, no distinction is made between upper and lower case characters. However, the values after the equals sign are transferred exactly as they are written in the file. Leading and trailing spaces are truncated.

5.1.1 Naming conventions

In the Physical Layer Configuration File and in the Application Layer Configuration File, several groups of keywords and sections are allowed. These refer to other sections and reflect the relationships and interconnections. The section names follow the naming conventions indicated below.

The section name begins with the section type followed by an arrow („->“, a minus sign followed by a greater than sign). A unique name appears after the arrow. No spaces are permitted between the name and the arrow.

In the section names, no distinction is made between upper and lower case characters.

The following characters are permitted for the logical names, the device names and the bench names.

„A“ ... „Z“	Upper case characters
„a“ ... „z“	Lower case characters
„0“ ... „9“	Numbers
„_“	Underscore
„.“	Decimal point

Table 5-1 Character set for names

The following maximum character lengths are permitted for section names, keywords and values:

section	80 characters
key	80 characters
value	260 characters

Table 5-2 Maximum character lengths

[LogicalNames]	This section contains a list of names of devices and benches. Any name can be used to identify a device or bench (Application Layer Configuration File).
[Device->...]	A device section contains different keywords to identify the devices. These include the GPIB address, the device type etc. (Physical Layer Configuration File and Application Layer Configuration File).
[Bench->...]	This section contains a group of device entries which together form a bench. A High Level Library requires the name of a bench in its setup routine (Application Layer Configuration File).
[ResourceManager]	This section contains information for the configuration of the Resource Manager.

5.1.2 [LogicalNames] section

The [LogicalNames] section is used to assign a short, meaningful name to a device or bench. Any name can be chosen. This section contains a list of unique name allocations. The values on the right side of the expressions must be valid names of a bench or of a device section.

The [LogicalNames] section is an optional entry and is used only in the

Application Layer Configuration File.

Example:

```
[LogicalNames]
CMD = bench->gsm
Power = device->keithley
```

5.1.3 [Device] section

The [Device] section contains a list of keywords and assigned values. These keywords and values precisely describe the relevant device. The name of the [Device] section begins with „Device->“ followed by a unique name. Any name can be chosen.

There must be a [Device] section for each device in the Physical Layer Configuration File.

A [Device] section with the same name can be defined in the Application Layer Configuration File. Additional device information can be given at this point by means of further keywords and values, or device information from the Physical Layer Configuration File can be overwritten. However, it is not possible to define a [Device] section in the Application Layer Configuration File which is not present in the Physical Layer Configuration File.

The keywords in a [Device] section and their meaning depend on the libraries used by the devices.

Keyword	Description
Description	*Optional entry* Device description, remarks
Type	*Mandatory entry* Device type (e.g. CMU etc.)
ResourceDesc	*Mandatory entry* VISA device properties and device description in the form: GPIB[card number]::[primary address]::[secondary address] PXI[segment number]::[device number]::[function]::INSTR Examples: „GPIB0::15“ or „PXI0::16::0::INSTR“
DriverSetup	*Optional entry* Special setup string for IVI driver, e.g. for simulation of devices

Table 5-3 Standard keywords of [Device] section

The „Type“ and „ResourceDesc“ entries are required for the test libraries (e.g. GSM.LIB). Both entries must be present in the Physical Layer Configuration File.

The information from the „Type“ entry allows the test libraries to distinguish between different supported devices (such as CMD55 or CMU). This information is also needed for the system self-test. The information from the „ResourceDesc“ entry is needed to set up the device driver and create the physical connection with the indicated device.

Example:

```
[device->CMD55]
Description = Radio Communication Tester CMD55
Type = CMD55
ResourceDesc = GPIB0::4
```

5.1.4 [Bench] section

The [Bench] section contains a list of keywords and assigned values which describes a group of devices and their use. The name of the [Bench] section begins with „Bench->“ followed by a unique name. Any name can be chosen. A [Bench] section can only be defined in the Application Layer Configuration File.

The keywords in a [Bench] section depend on the test library used by the bench. A keyword always provides at least one reference to a device entry. Other keywords may be necessary to describe the bench. The following keywords are predefined and should be present in each [Bench] section.

Keyword	Description
Description	*Optional entry* Bench description, remarks
Simulation	*Optional entry* If set to „1“, the complete bench is simulated by the test library.
Trace	*Optional entry* If set to „1“, the tracing function is enabled for the test library

Table 5-4 Standard keywords of [Bench] section

The [Bench] section can contain further useful keywords and values which are used by a test library. For the test library GSM.LIB, entries for the calibration, the calibration files, the network etc. may be useful.

Example:

```
[bench->GSM]
RadioComTester = device->CMD55
UUT = device->COM2
Simulation = 0
Trace = 0
```

5.1.5 [ResourceManager] section

The [ResourceManager] section contains keywords and assigned values to control the behaviour of the Resource Manager library. The following keywords are supported:

Key name	Remarks
Trace	Blocks the tracing function (value = 0), enables the tracing function (value = 1). The function impacts on all libraries.
TraceFile	Defines the path and the name of the trace file.
TraceToScreen	The tracing information is displayed on the standard screen (value = 1).
TraceTimeStamp	Writes the time of day at the start of each tracing line (value = 1).
TraceThreadID	Writes the ID of the current thread at the start of each tracing line (value = 1).

Table 5-5 Keywords of [ResourceManager] section

5.2 PHYSICAL.INI

In the file PHYSICAL.INI (Physical Layer Configuration File), all hardware assemblies available in the Generic Test Software Library are described along with the corresponding definitions and settings (see example PHYSICAL.INI file). This file also contains definitions which are applicable to all test applications to be executed on the system (e.g. type definition). The information entered in this file is used by all test libraries and thus by each test step.

The PHYSICAL.INI file normally exists only once in the system as it reflects the exact physical structure. The file must only be modified in the event of a hardware change.

The Resource Manager calls and administers the information from the PHYSICAL.INI file.

5.2.1 Example file for PHYSICAL.INI

```
                                                                   Description
[device->CMD80]                                                        1
Description = 'Radio Communication Tester CMD80'                       2
Type = CMD80                                                           3
ResourceDesc = GPIB1::1                                                4

[Device->CMD55]                                                        1
description = 'Radio Com Tester CMD55'                                 2
Type = CMD55                                                           3
ResourceDesc = GPIB0::1                                                4

[Device->CMU]                                                          1
description = 'Radio Com Tester CMU'                                   2
Type = CMU                                                             3
ResourceDesc = GPIB0::20                                               4
; Networks                                                             5
ResourceDesc_RF_NSig       = GPIB0::20::1                              6
ResourceDesc_GSM_900NSig   = GPIB0::20::2                              6
ResourceDesc_GSM_900Sig    = GPIB0::20::3                              6
ResourceDesc_GSM_1800NSig  = GPIB0::20::4                              6
ResourceDesc_GSM_1800Sig   = GPIB0::20::5                              6
ResourceDesc_GSM_1900NSig  = GPIB0::20::6                              6
ResourceDesc_GSM_1900Sig   = GPIB0::20::7                              6
ResourceDesc_Audio         = GPIB0::20::8                              6

[Device->MOBILE]                                                       1
Type = MOBILE_XYZ                                                      3
ResourceDesc = COM3                                                    4
```

```
                                                          Description
[device->PSU1]                                                 1
Description = 'Power Supply Keithley 2306'                     2
Type = KI2306                                                  3
ResourceDesc = GPIB0::16                                       4

[device->PSU2]                                                 1
Description = 'Power Supply Keithley 2306'                     2
Type = KI2306                                                  3
ResourceDesc = GPIB1::16                                       4

[device->RelayCard1]                                           1
Description  = 'Relay Card 1'                                  2
Type         = PRL1                                            3
ResourceDesc = PXI1::10::0::INSTR                              4
DriverPrefix = rsprl1                                          7
DriverDll    = rsprl1.dll                                      8
DriverOption = 'Simulate=0,RangeCheck=1'                       9

; mantadory analog bus entry                                   1
[device->ABUS]                                                 2
Description = 'Analog Bus'                                     3
type = ab                                                      4

; hard wired connections
[io_channel->system]                                           10

Battery_Force   = RelayCard1!HiPwr1com                         11
Battery_Sense   = RelayCard1!Pwr1com                           11
Charger_Force   = RelayCard1!Pwr2com                           11
Charger_Sense   = RelayCard1!Pwr3com                           11

RCT_AF_In_P   = RelayCard1!RB1com                              11
RCT_AF_In_N   = RelayCard1!RB2com                              11
RCT_AF_In_P.1 = RelayCard1!RB7no                               11
RCT_AF_In_N.1 = RelayCard1!RB8no                               11
RCT_AF_Out_P  = RelayCard1!RB3com                              11
RCT_AF_Out_N  = RelayCard1!RB4com                              11
RCT_AF_Out_P.1  = RelayCard1!RB5com                            11
RCT_AF_Out_N.1  = RelayCard1!RB6com                            11
```

5.2.2 Description of example file PHYSICAL.INI

The description is based on the example file in section 5.2.1. The indicated numbers refer to the corresponding positions in the example file. The place-holder „XY“ in the following listing stands for the corresponding entries.

1	`[device->XY]`	Defines the name under which the device is called in the test libraries. A separate entry must be made for each device. The entry in square brackets [ ] defines a new section within which new definitions are made.
2	`Description = „XY“`	Gives a detailed description of the defined device. The entry is optional.
3	`Type = „XY“`	Gives the exact designation of the defined device. This designation is needed to call the corresponding device driver. The entry is mandatory.
4	`ResourceDesc = „XY“`	Gives the necessary hardware information required for the defined device. The entry is mandatory. Details provided at this point include, for example: GPIB address: GPIB1::20::1 (example) GPIB[card number]::[primary address]::[secondary address] Serial interface: COMX PXI address: PXI1::10::0::INSTR (example) PXI[segment number]::[device number]::[function]::INSTR
5		Text appearing after a semicolon (;) is interpreted as a comment.
6	`ResourceDesc_XY =`	In the case of certain devices, special subassemblies can be addressed directly through their primary address and secondary address. The relevant hardware information can be indicated especially for this subassembly with the corresponding designation.
7	`DriverPrefix = XY`	Gives a prefix for the device driver.
8	`DriverDll = XY`	Gives the path and the file name of the device driver.
9	`Driver Option = XY`	Gives certain options applicable to the device driver.
10 + 11	`[IO_Channel->system]`	The following definitions of I/O channels apply to all applications executed on the system. On the right side are the physical channel names as defined by the hardware and by the device driver. On the left side are the logical channel names as used in the test libraries.

Table 5-6 Description of PHYSICAL.INI

5.3 APPLICATION.INI

In the APPLICATION.INI file (Application Layer Configuration File) is a description of how the individual test libraries and the test functions use the hardware components (see example file APPLICATION.INI). Different hardware components can be combined into groups (bench). This bench can then be used within the test function. Furthermore, definitions are made in this file which apply to certain test applications to be executed on the system (e.g. definition of designations in the case of multi-channel operation).

The Resource Manager calls and administers the information from the APPLICATION.INI file.

NOTE:

Since an Application Layer Configuration File (APPLICATION.INI) normally exists for each test application executed on the system, the file name can be matched to the test application in question, e.g. APP_XXX.INI. The Resource Manager is told during setup which Application Layer Configuration File is to be used.

For ease of comprehension, the file name APPLICATION.INI is used for the Application Layer Configuration File in the manual.

5.3.1 Example file for APPLICATION.INI

```
                                                                  Description
[ResourceManager]                                                       1
Trace=1                                                                 2
TraceFile = c:\Temp\Trace.log                                           3

[LogicalNames]                                                          4
GSM = bench->gsm                                                        5
CdmaAnag = bench->CdmaAnag                                              6

[bench->gsm]                                                            7
PowerSupply = device->PSU1                                              8
;RadioComTester = device->CMD55                                         8
RadioComTester = device->CMU                                            8
UUT = device->MOBILE                                                    8
Trace = 1                                                               2
```

```
                                                                   Description
Calibration = 1                                                         9
CalibrationFile  = c:\GTSL\Configuration\TestCalGSM.cal                10
CalibrationFile2 = c:\GTSL\Configuration\TestCalAUDIO.cal              10
CalibrationPath_GSM_Rx= tableGSMRx                                     11
CalibrationPath_GSM_Tx= tableGSMTx                                     11

CalibrationPath_AUDIO_Ear= tableAUDIO1,tableAUDIO2                     11
CalibrationPath_AUDIO_Mouth= tableAUDIO1,tableAUDIO2                   11

SwitchDevice1 = device->RelayCard1                                     12
AppChannelTable = io_channel->BenchGSM                                 12
AnalogBus = device->ABUS                                               12

[io_channel->BenchGSM]                                                 13

MS_Battery_Force = RelayCard1!HiPwr1no                                 14
MS_Battery_Sense = RelayCard1!Pwr1no                                   14
MS_Charger_Force = RelayCard1!Pwr2no                                   14
MS_Charger_Sense = RelayCard1!Pwr3no                                   14

MS_AF_Ext_Out_P = RelayCard1!RB1no                                     14
MS_AF_Ext_Out_N = RelayCard1!RB2no                                     14
MS_AF_Ext_In_P  = RelayCard1!RB3no                                     14
MS_AF_Ext_In_N  = RelayCard1!RB4no                                     14

Fixture_Mouth_P = RelayCard1!RB5no                                     14
Fixture_Mouth_N = RelayCard1!RB6no                                     14

Fixture_Ear_P   = RelayCard1!RB7com                                    14
Fixture_Ear_M   = RelayCard1!RB8com                                    14

MS_Car_Kit_Det      = RelayCard1!RA1no                                 14
MS_Car_Kit_Det_Com  = RelayCard1!RA1com                                14

Fixture_Box_Lock   = RelayCard1!RA2no                                  14
VCC24.1   = RelayCard1!RA2com                                          14
Fixture_RF_Relay   = RelayCard1!RA3no                                  14
VCC12.1   = RelayCard1!RA3com                                          14
Fixture_SIM_Con    = RelayCard1!RA4no                                  14
VCC24.2   = RelayCard1!RA4com                                          14
Fixture_RF_Con     = RelayCard1!RA5no                                  14
VCC24.3   = RelayCard1!RA5com                                          14

[bench->CdmaAnag]                                                       7
PowerSupply = device->PSU2                                              8
RadioComTester = device->CMD80                                          8
Trace = 1                                                               2
```

```
                                                                    Description
Calibration = 1                                                          9

CalibrationFile  = c:\GTSL\Configuration\TestCalCDMA.cal
CalibrationFile2 = c:\GTSL\Configuration\TestCalANAG.cal

CalibrationPath_CDMA_Rx= tableCDMARx                                    11
CalibrationPath_ANAG_Rx= tableANAGRx                                    11

CalibrationPath_CDMA_Tx= tableCDMATx                                    11
CalibrationPath_ANAG_Tx= tableCDMARx                                    11
```

5.3.2 Description of example file APPLICATION.INI

The description is based on the example file in section 5.3.1. The indicated numbers refer to the corresponding positions in the example file. The place-holder „XY“ in the following listing stands for the corresponding entries.

1	`[ResourceManager]`	Defines a new section (identified by the square brackets [ ]) with information evaluated directly by the Resource Manager.

2	`Trace = 0`
	`Trace = 1`
		Tracing of information is not carried out (Trace = 0) or is carried out (Trace = 1).

3	`TraceFile = XY`	Gives the path and file name for storing trace information.
4 to 6	`[LogicalNames]`	Defines a new section in which logical short names are defined. The short names can be used to call the libraries.
7	`[bench->XY]`	Defines a new bench with its name. The name, which is defined at this point, is called in the SETUP routine of the corresponding test library.
8		The listed devices with the relevant defined names are assigned to the bench. The addressed devices must be defined in the PHYSICAL.INI file with their details.

9	`Calibration = 0`
	`Calibration = 1`
		When carrying out measurements with the devices defined in the bench, correction of the measured values (calibration=1) must be carried out with the values indicated in the „Calibration Files“.

10	`CalibrationFile = XY`
	`CalibrationFile<i> = XY`
		Gives the path and the file name of the first „Calibration Files“. Gives the path and the file name of the further „Calibration Files“, <i> being a number from 2 ... n.

Table 5-7 Description of APPLICATION.INI

11	`CalibrationPath = XY`	Gives the name of a correction table from the „Calibration File“. A „Calibration File“ contains several correction tables.
12		Defines logical names for use in the Switch Manager library.
13 + 14	`[io_channel->XY]`	The following definitions of I/O channels apply to the applications executed with the indicated bench. On the right side are the physical channel names as defined by the hardware and by the device driver. On the left side are the logical channel names as used in the test libraries.

Table 5-7 Description of APPLICATION.INI

6 Editing and running test sequences

6.1 TestStand

6.1.1 General

The test sequences created by ROHDE & SCHWARZ are edited under the TestStand Sequence Editor from National Instruments.

This section provides just a brief description of how a test sequence is edited and run.

NOTE:

A full description of the operation and functions of the individual menu windows of the TestStand Sequence Editor will be found in the enclosed User Manual or the online documentation.

A test sequence is edited as follows:

Sequence Editor

1. Open TestStand Sequence Editor with **Start -> Programs -> National Instruments TestStand -> Sequence Editor** or by clicking the **Sequence Editor** icon on the Windows 2000/XP desktop. You will now see the main screen of the Sequence Editor. Enter your user name and password (see Figure 6-1). The user name and password are stored in the user profiles of the TestStand software.

   Default setting:

User Name:	administrator
Password:	no password needed

17th Issue 08.09

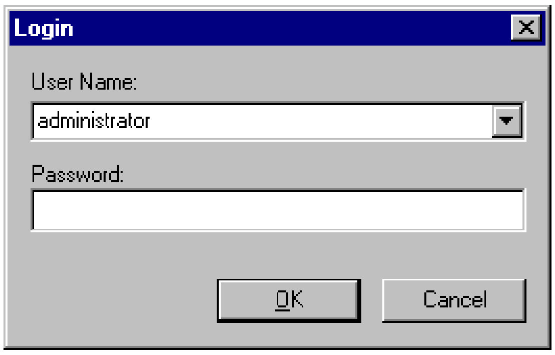
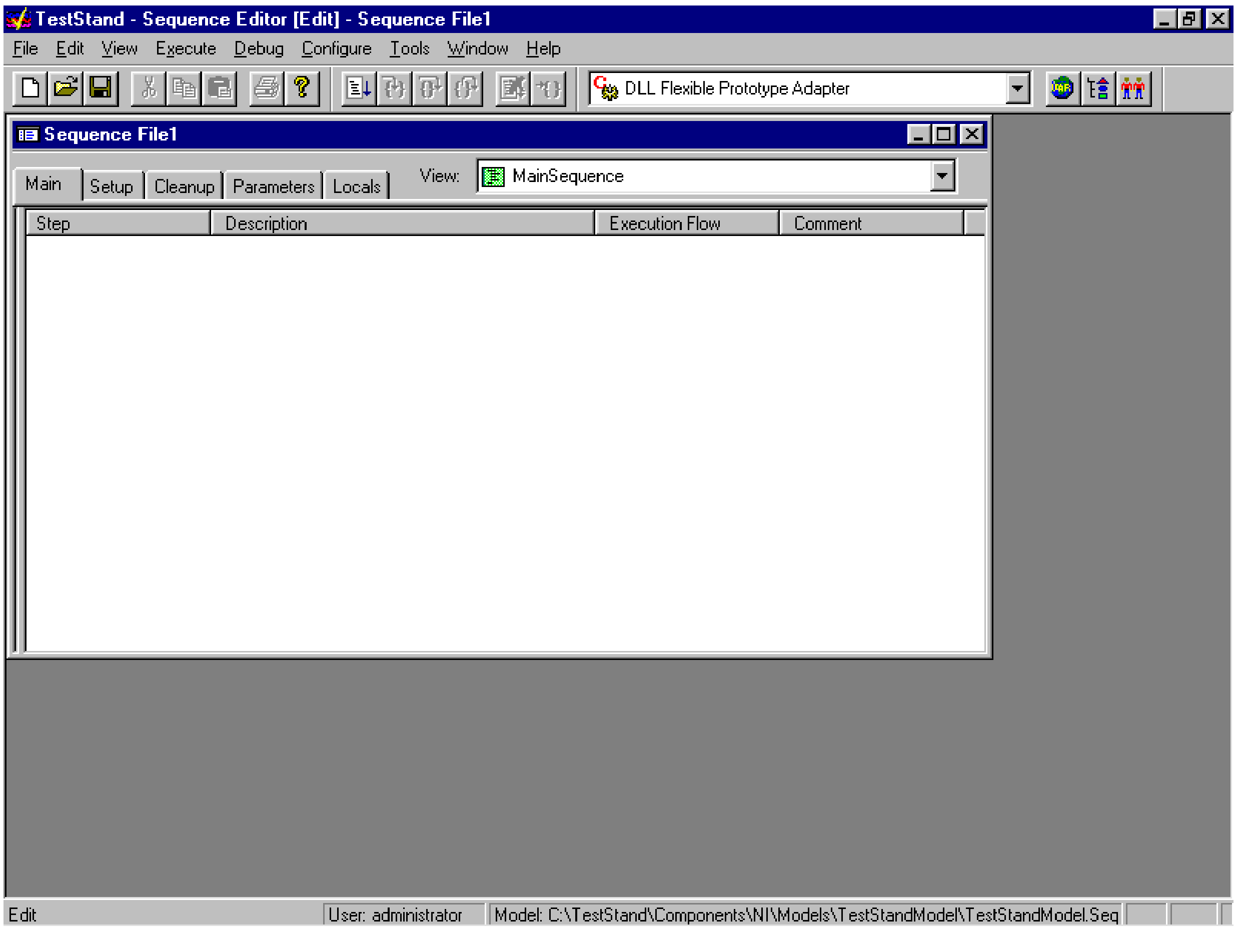


Figure 6-1 Login

NOTE:

User name and password can be set according to your specific company requirements.


TestStand - Sequence Editor [Edit] - Sequence File1
File Edit View Execute Debug Configure Tools Window Help
DLL Flexible Prototype Adapter
Sequence File1
Main Setup Cleanup Parameters Locals
View: MainSequence
Step Description Execution Flow Comment
Edit

User: administrator

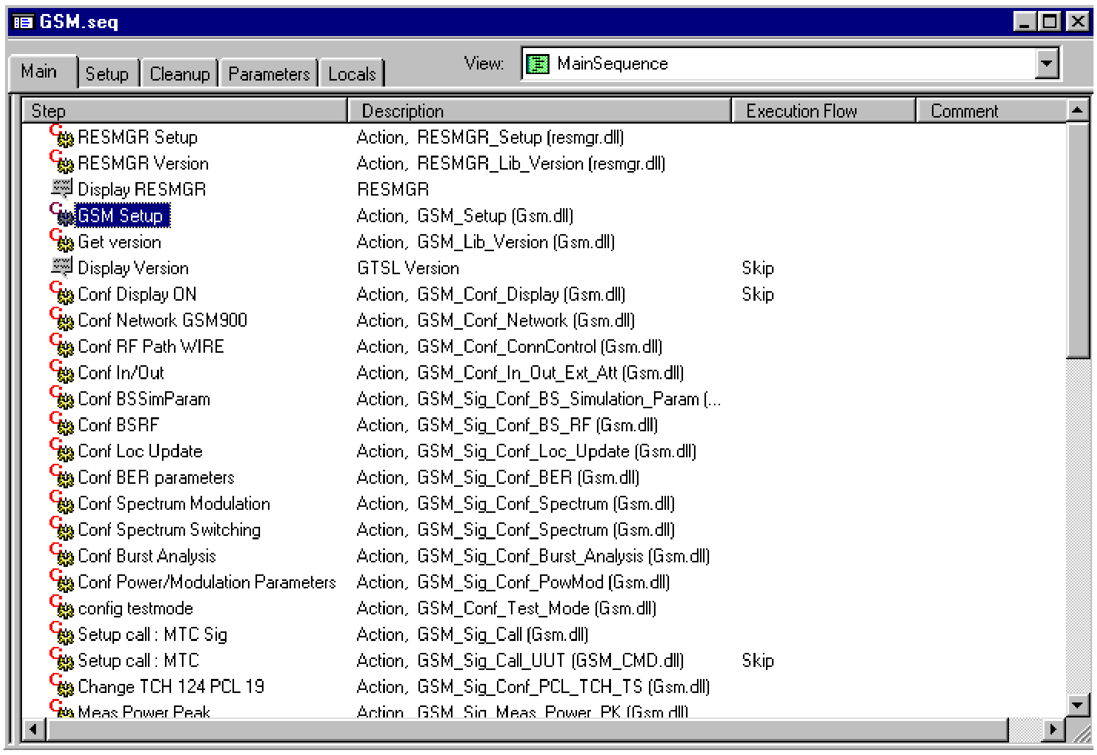
Model: C:\TestStand\Components\NI\Models\TestStandModel\TestStandModel.Seq

Figure 6-2 TestStand main screen

2. Open an existing test sequence with <File><Open...>.

   The test sequences created and supplied by ROHDE & SCHWARZ are stored by default in the directory `C:\Program Files\GTSL\Sequences`. If a different directory was specified during the software installation, the test sequences will be stored there (`...\Sequences`).

   The opened test sequence and its steps are displayed in the working window.



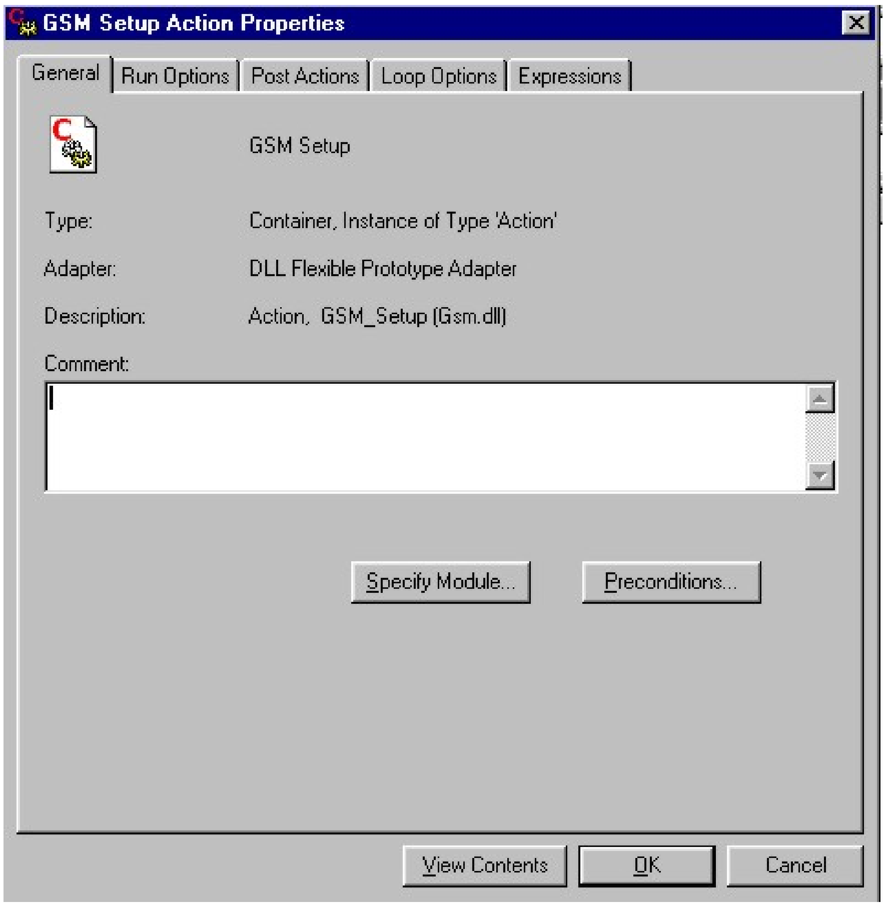
Step	Description	Execution Flow	Comment
RESMGR Setup	Action, RESMGR_Setup (resmgr.dll)		
RESMGR Version	Action, RESMGR_Lib_Version (resmgr.dll)		
Display RESMGR	RESMGR		
GSM Setup	Action, GSM_Setup (Gsm.dll)		
Get version	Action, GSM_Lib_Version (Gsm.dll)		
Display Version	GTSL Version	Skip	
Conf Display ON	Action, GSM_Conf_Display (Gsm.dll)	Skip	
Conf Network GSM900	Action, GSM_Conf_Network (Gsm.dll)		
Conf RF Path WIRE	Action, GSM_Conf_ConnControl (Gsm.dll)		
Conf In/Out	Action, GSM_Conf_In_Out_Ext_Att (Gsm.dll)		
Conf BSSimParam	Action, GSM_Sig_Conf_BS_Simulation_Param (...		
Conf BSRF	Action, GSM_Sig_Conf_BS_RF (Gsm.dll)		
Conf Loc Update	Action, GSM_Sig_Conf_Loc_Update (Gsm.dll)		
Conf BER parameters	Action, GSM_Sig_Conf_BER (Gsm.dll)		
Conf Spectrum Modulation	Action, GSM_Sig_Conf_Spectrum (Gsm.dll)		
Conf Spectrum Switching	Action, GSM_Sig_Conf_Spectrum (Gsm.dll)		
Conf Burst Analysis	Action, GSM_Sig_Conf_Burst_Analysis (Gsm.dll)		
Conf Power/Modulation Parameters	Action, GSM_Sig_Conf_PowMod (Gsm.dll)		
config testmode	Action, GSM_Conf_Test_Mode (Gsm.dll)		
Setup call : MTC Sig	Action, GSM_Sig_Call (Gsm.dll)		
Setup call : MTC	Action, GSM_Sig_Call_UUT (GSM_CMD.dll)	Skip	
Change TCH 124 PCL 19	Action, GSM_Sig_Conf_PCL_TCH_TS (Gsm.dll)		
Meas Power Peak	Action, GSM_Sig_Meas_Power_PK (Gsm.dll)		

Figure 6-3 Working window

6.1.2 Editing a test step

Edit an individual test step as follows:

1. Double click the test step you wish to edit in the working window. This opens the **Test Step Setup** menu window.



GSM Setup Action Properties

GSM Setup

Type: Container, Instance of Type 'Action'

Adapter: DLL Flexible Prototype Adapter

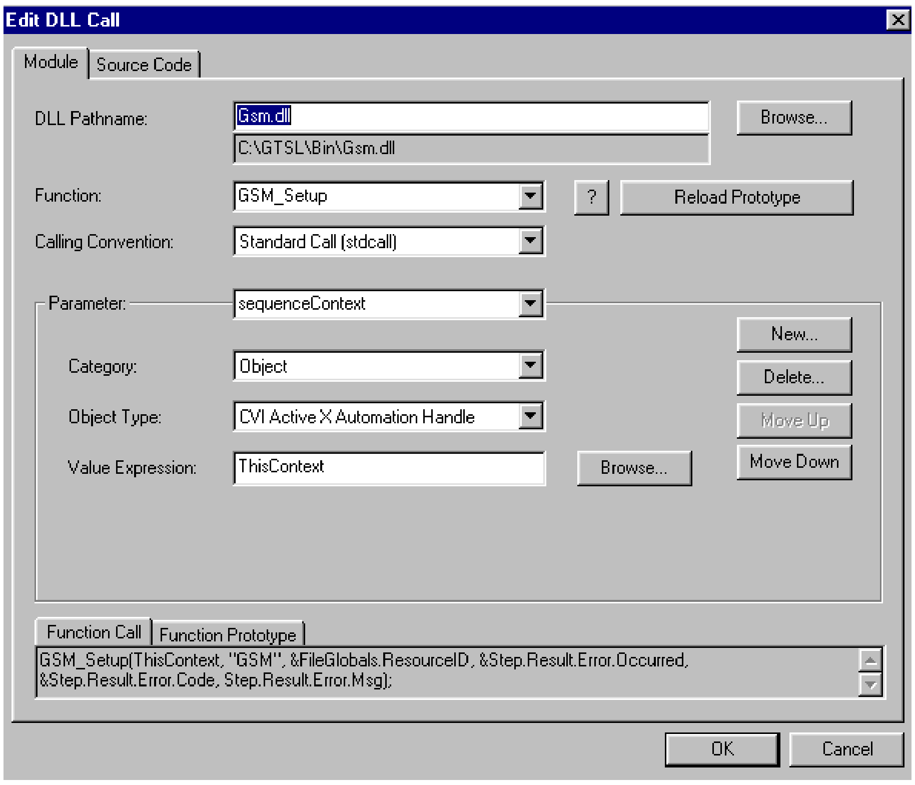
Description: Action, GSM_Setup (Gsm.dll)

Comment:

Figure 6-4 Test step setup

Settings for the selected test step are made in this window. The name of the step and the type of call are shown in the title bar.

2. Click **Specify Module** to open the **Edit DLL Call** menu window for editing the test step.



DLL Pathname: Gsm.dll
C:\GTSL\Bin\Gsm.dll
Function: GSM_Setup
Calling Convention: Standard Call (stdcall)
Parameter: sequenceContext
Category: Object
Object Type: CVI ActiveX Automation Handle
Value Expression: ThisContext

```
GSM_Setup(ThisContext, "GSM", &FileGlobals.ResourceID, &Step.Result.Error.Occurred,
&Step.Result.Error.Code, Step.Result.Error.Msg);
```

Figure 6-5 Edit DLL Call

The settings for the selected test step are made on the **Module** tab in this window.

DLL Pathname:	Shows the name of the test library or the name of the dynamic link library (DLL) of the test library. The memory path of the DLL file is displayed.
Browse...	Opens the directory structure for the selection of the test library (DLL)
Function	Shows a pick list of the test functions in the test library. The selected test function is displayed.
?	Shows the help text for the selected test function. The function, purpose and parameters of the test function are displayed.
Reload Prototype	The test function is reloaded
Calling Convention:	Always set to Standard Call in the case of R&S test functions
Parameter:	Displays a pick list of the parameters contained in the test function. The selected parameter is displayed.

Category:	Shows the various parameter settings. These settings are parameter specific.
Object Type	
Value Expression	
Browse...	
New..	These buttons are used to insert, delete and move parameters manually.
Delete...	
Move Up	
Move Down	

6.1.3 Running test sequences

The edited test sequence can be run directly in the TestStand Sequence Editor.

The opened test sequence is run once with <Execute><Single Pass>. The started test sequence is run in a separate window. The result of the test sequence is displayed in a Report Window.

Select <Execute><Test UUTs> to run the open test sequence in a continuous loop. The operator is prompted to enter a serial number before each run. The started test sequence is run in a separate window. Clicking **Stop** in the serial number input window cancels the test sequences. The results of the test sequences are displayed in a Report Window with the corresponding serial number.

The Report Window must be closed when the test sequences have completed.

6.2 Generic Test Operator Interface R&S GTOP

6.2.1 General

The generated test sequences can also be executed via the Generic Test Operator Interface R&S GTOP. The R&S GTOP is a user interface, which has been specifically developed for application in multi-channel systems in the production area. R&S GTOP can only be used to run test sequences. Test sequences cannot be created or altered. Neither is it possible to debug test sequences.

The TestStand Run Time Engine and the Operator Interface Library are required to run the test sequences in the Generic Test Operator Interface (R&S GTOP). Figure 6-6 shows the integration of the R&S GTOP when performing a test sequence.

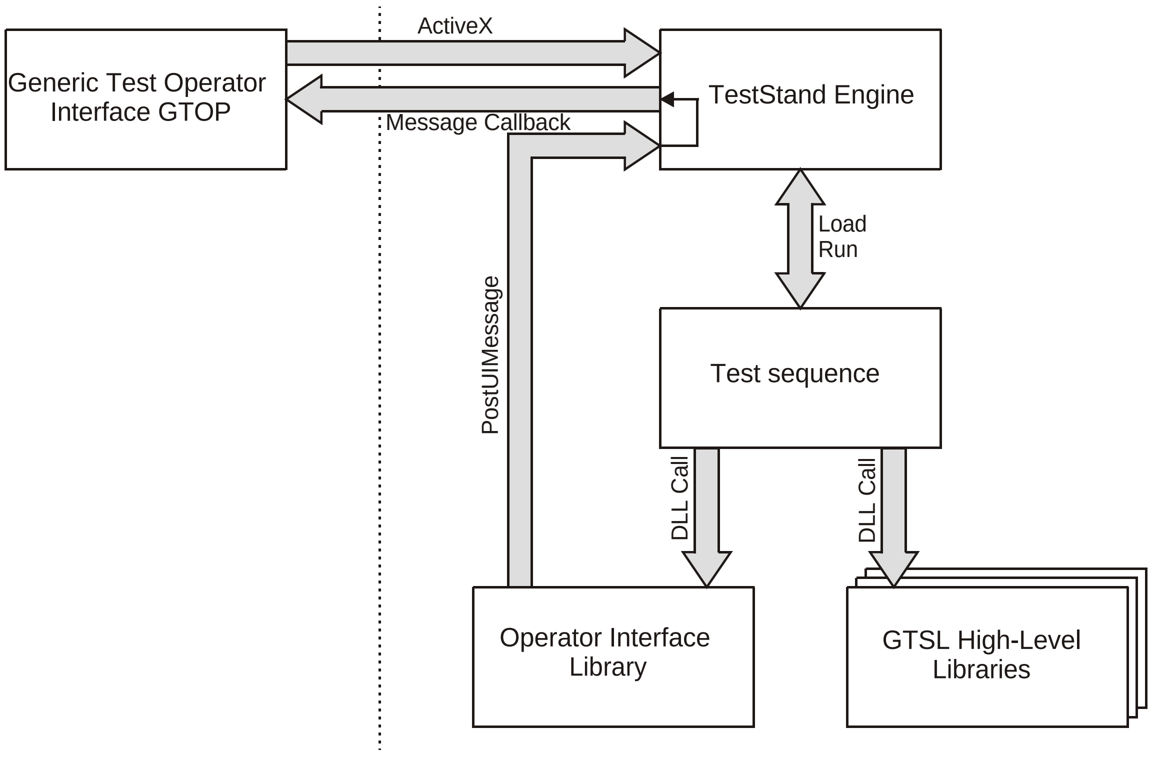


Figure 6-6 Integration of Generic Test Operator Interface R&S GTOP

The Operator Interface Library provides the functions for interaction between the test sequence in the TestStand and R&S GTOP.

NOTE:

The individual functions in the Operator Interface Library are described in the enclosed help file.

The directory `...\gtsl\operatorinterface\develop\gtop` contains several example files, which renders the configuration and the running of R&S GTOP apparent. These files include:

- `gtop_demo.bat`
  shows how to call R&S GTOP from the command line.
- `gtop_demo.ini`
  is a sample configuration file for R&S GTOP
- `gtop_demo.seq`
  is a channel-independent sequence which shows how to
  - use the OPERINT functions in the PreUUT and PostUUT callbacks
  - use OPERINT_Get_Channel to find out if R&S GTOP is available and on which channel the sequence is running.
  - make a sequence independent from the operator interface, i.e. how it can run with optimum results on R&S GTOP and with good results on a different operator interface like the TestStand sequence editor.
- `gtop_demo_phys.ini, gtop_demo_appl.ini`
  The files are the physical and application layer ini files for the resource manager

Although R&S GTOP has been designed to meet many wishes of our customers, you may want to customize the appearance or functionality of the operator interface. Therefore, the R&S GTOP application and the OPERINT library are shipped with source code. You will find the sources in the following directories:

- `...\gtsl\operatorinterface\develop\gtop`
- `...\gtsl\develop\libraries\operint`

NOTE:

Do not modify the original source files. Copy all files to a different directory outside the R&S GTSL directory tree and work on the copy, otherwise your changes may be overwritten after the next R&S GTSL update.

How to modify the appearance of R&S GTOP:

Load the project file `GTOP.PRJ` in CVI and open the file `GTOP.UIR`. Modify text and colors, include your company logo etc. as required. However, be careful with modifications of size and position of any panel.

How to modify the functionality of R&S GTOP:

Load the project file `GTOP.PRJ` in CVI and modify the source code. The R&S GTOP project is based on the National Instruments CVI operator interface, which is part of TestStand. Once you understand how the CVI operator interface works, you will also be able to modify the functionality of R&S GTOP.

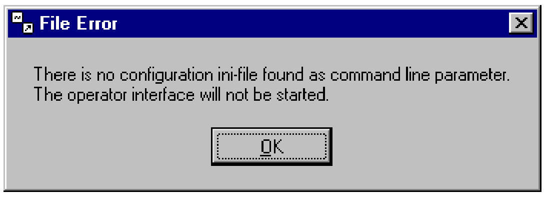
Most of the R&S GTOP specific functions can be found in the source module `applspec.c`.

6.2.2 Running R&S GTOP

The Generic Test Operator Interface R&S GTOP is run using the `gtop.exe` file. When running the file a R&S GTOP configuration file name must also be given (see 6.2.4: R&S GTOP Configuration File).

Example: `gtop.exe gtop_demo.ini`

If, when run, a R&S GTOP configuration file is not specified, an error message is issued and the start of the program is canceled.



There is no configuration ini-file found as command line parameter.
The operator interface will not be started.

Figure 6-7 Start Error Message

If a valid R&S GTOP configuration file is present, R&S GTOP is started with the details specified within it. The start screen is displayed (see Figure 6-8) and a request to enter the user name and password (see Figure 6-9). The user name and password are specified in the Test-Stand software's user profile.

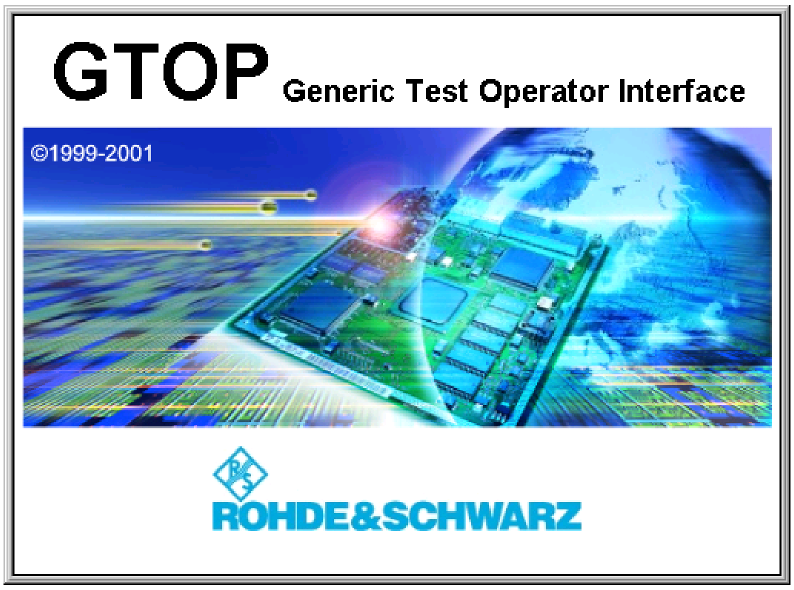


Figure 6-8 Start Screen

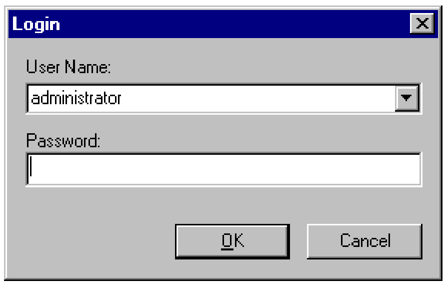


Figure 6-9 Login

Default Configuration

User Name:	administrator
Password:	No password required

NOTE:

User name and password can be defined in company specific terms.

The exact procedure for defining a user name and password is described in the enclosed TestStand documentation.

After a valid user name and password is entered the test sequences contained in the R&S GTOP configuration file are loaded and automatically started. It is not possible to stop individual test sequences. It is only possible to terminate the complete Generic Test Operator Interface R&S GTOP using the <file><Exit> menu item

6.2.3 Operator Interface

NOTE:

The Generic Test Operator Interface R&S GTOP shown below is configured for a 2-channel system. If a 1-channel system configuration is used, the right-hand operator interface remains blank.

6.2.3.1 Representation

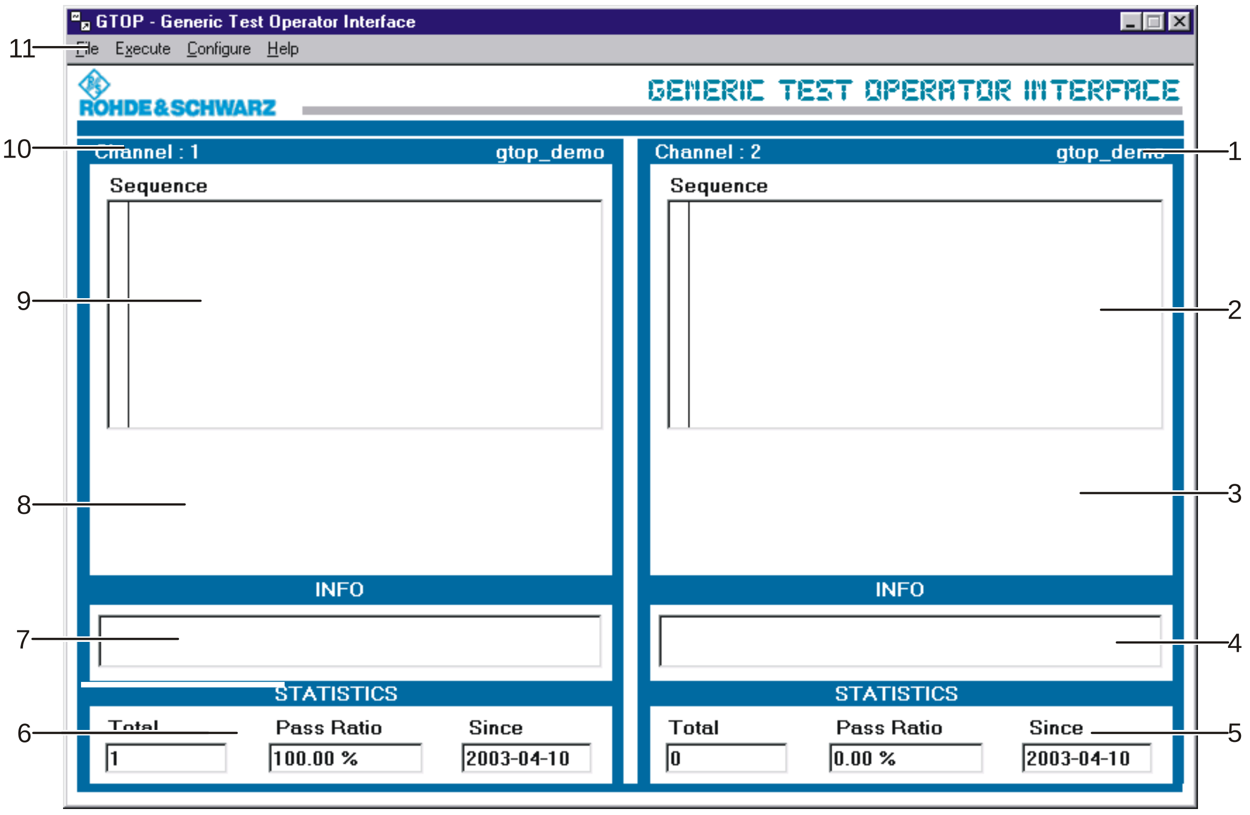


Figure 6-10 Generic Test Operator Interface R&S GTOP

1	Display panel, channel 2
2	Test-sequence display, channel 2
3	Banner display, channel 2
4	Information line, channel 2
5	Statistic line, channel 2
6	Statistic line, channel 1
7	Information line, channel 1
8	Banner display, channel 1
9	Test-sequence display, channel 1
10	Display panel, channel 1
11	Menu bar

6.2.3.2 Menu Bar

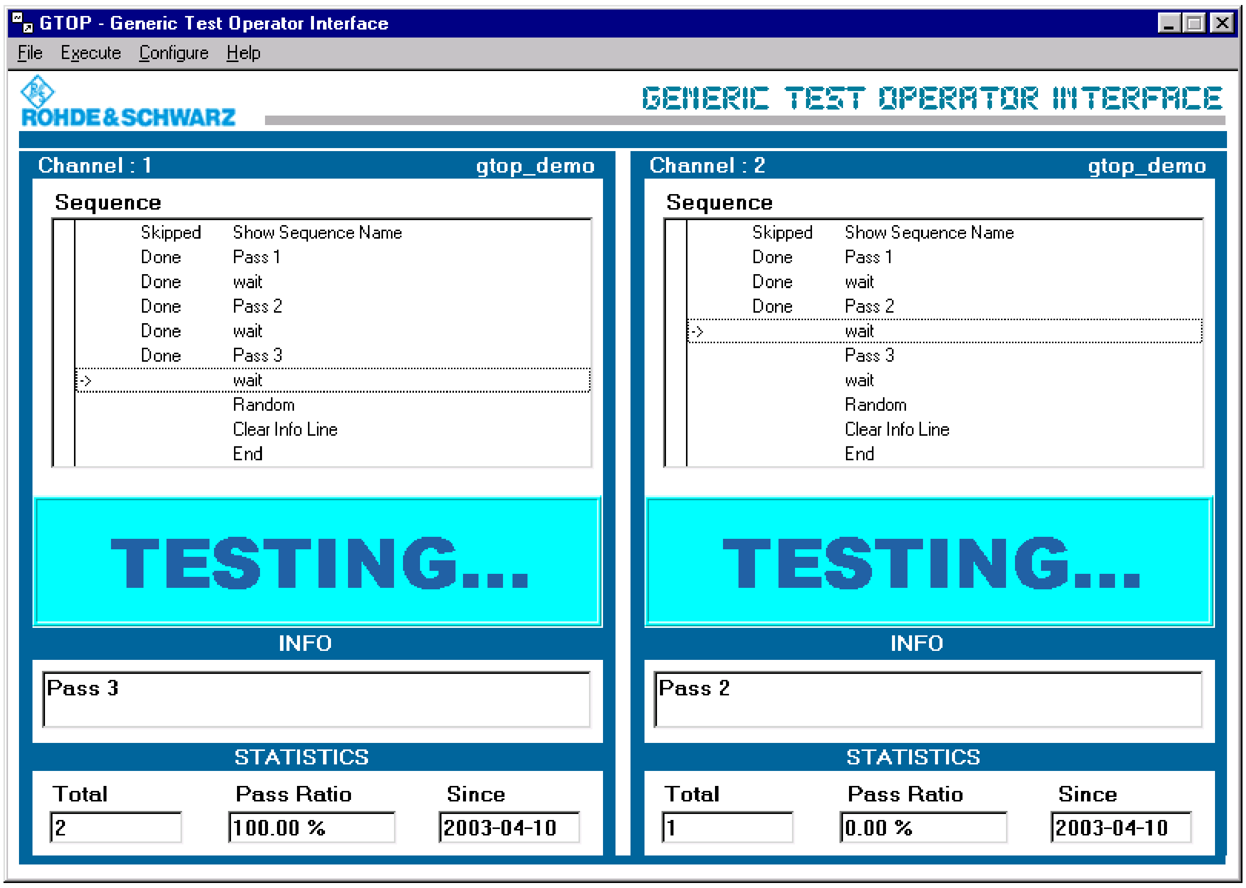
The menu bar is used to call up the functions for operating and configuring the Generic Test Operator Interface R&S GTOP. The menu items called up from R&S GTOP then open the corresponding dialogs in TestStand. The menu items are enabled or blocked in accordance with the user’s privileges (see also the TestStand documentation).

The table below describes the available functions.

File	Login	Login using another name (shift change, administrator login)
	Exit	Terminate all sequences, exit Generic Test Operator Interface R&S GTOP
Execute	Tracing Enabled	Switch tracing on and off in the test sequence display
Configure	Adapters	Default TestStand configuration dialog
	Station Options	Default TestStand configuration dialog
	External Viewers	Default TestStand configuration dialog
	Search Directories	Default TestStand configuration dialog
	Statistic Options	Configuration of statistic options (see section: 6.2.3.6)
	Report Options	Default TestStand configuration dialog
	Database Options	Default TestStand configuration dialog
Help	About	Display information on Generic Test Operator Interface R&S GTOP (version number)

Table 6-1 Menu Bar

6.2.3.3 Test Sequence Display



Channel : 1 — Sequence:

```
        Skipped   Show Sequence Name
        Done      Pass 1
        Done      wait
        Done      Pass 2
        Done      wait
        Done      Pass 3
->                wait
                  Random
                  Clear Info Line
                  End
```

INFO: Pass 3

Total	Pass Ratio	Since
2	100.00 %	2003-04-10

Channel : 2 — Sequence:

```
        Skipped   Show Sequence Name
        Done      Pass 1
        Done      wait
        Done      Pass 2
->                wait
                  Pass 3
                  wait
                  Random
                  Clear Info Line
                  End
```

INFO: Pass 2

Total	Pass Ratio	Since
1	0.00 %	2003-04-10

Figure 6-11 Test Sequence Display

The test sequence display represents a small section of the current test sequence. If under the <Execute> <Tracing Enabled> menu item, tracing is activated (ticked off), the current step is marked by an arrow „->“. Step flags are displayed by letters (e.g. „S“ for skip). This is followed by the step result (Done, Passed, Failed,..) and the name of the step. Faulty steps are displayed in another color (red).

6.2.3.4 Banner

Banners are changing, colored highlighted displays. Certain banners may cover up the test sequence display. The various banners are filed in the Operator Interface Library as functions and can be integrated into the test sequence.

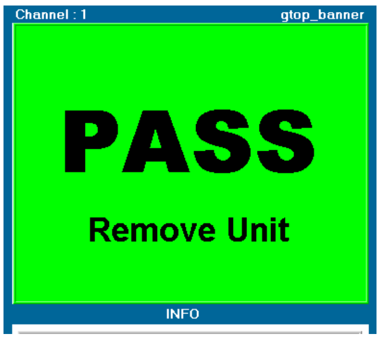
During a standard process, each channel of the test sequence process and the „Testing...“ banner below are displayed. Invoking the corresponding function from the Operator Interface Library enables the selected banner to be superimposed over the test sequence and thus displayed. The selected banner is displayed until

- the display of a new banner is invoked from the test sequence.
- the function for clearing the selected banner is invoked from the test sequence.

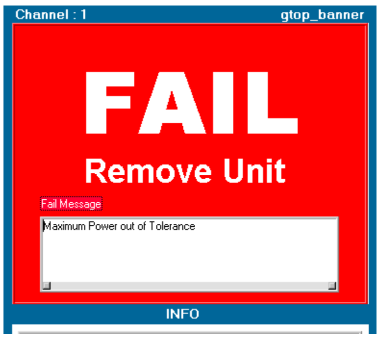
Banners are not placed on top of each other. Only one banner is displayed per channel. It is therefore not necessary to explicitly delete every single banner. Only when the test sequence display is to be shown again, is it necessary to delete the displayed banner.

There are two types of banner:

1. Permanent banners

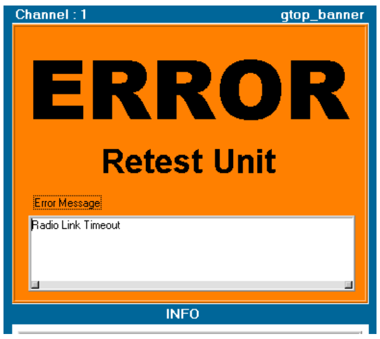


Pass Banner



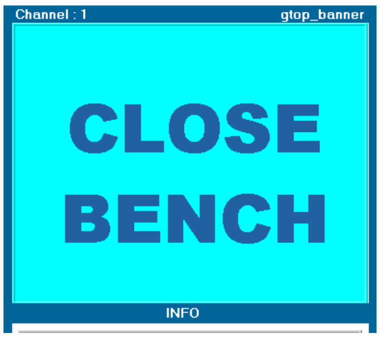
Maximum Power out of Tolerance

Fail Banner

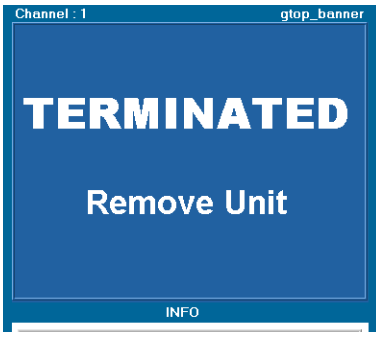


Radio Link Timeout

Error Banner



Close Bench Banner

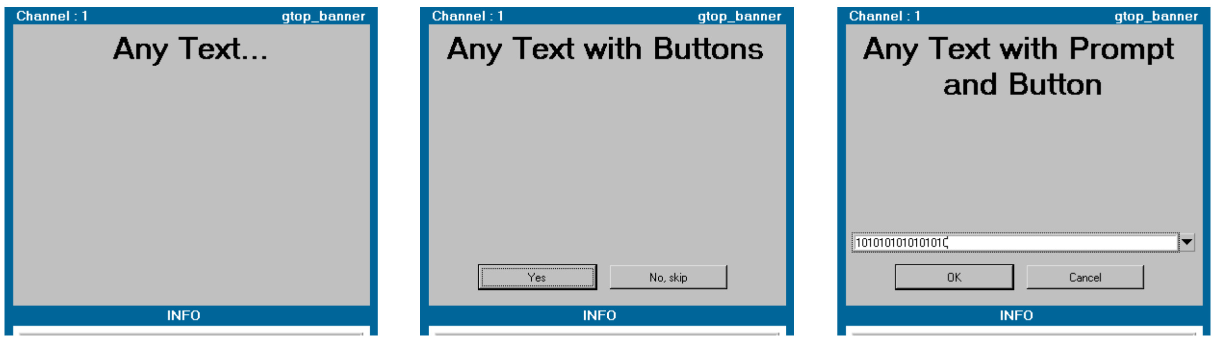


Terminated Banner

Figure 6-12 Permanent banners

Permanent banner text is given in the appropriate function of the Operator Interface Library. The texts can be easily converted to UIR files without any modification to the source code. On the „Fail Banner“ and „Error Banner“ additional free text can also be displayed. The text is transfered from the test sequence to the banner. See also the help file in the Operator Interface Library.

2. Configurable banners



Text Banner Dialog Banner Dialog and Prompt Banner

Figure 6-13 Configurable Banners

The content of the configurable banners is transferred over the corresponding Operator Interface Library functions from the test sequence to the Generic Test Operator Interface R&S GTOP. The following configurations are possible:

- Text content
- Button inscription
- Text and background color

The „Text Banner“ displays a freely selectable text. The „Text Banner“ is displayed until a new banner is invoked or the banner displayed is deleted.

The „Dialog Banner“ displays a freely selectable text and up to two buttons. The „Dialog Banner“ is displayed until a button is clicked.

The „Dialog and Prompt Banner“ displays a freely selectable text and an input capability (e.g. for entering a serial number). Here too, up to two buttons can be displayed. The data entered is returned to the test sequences. The „Dialog and Prompt Banner “ is displayed until a button is clicked.

See also the help file in the Operator Interface Library.

6.2.3.5 Information Bar

The user can display freely selectable text on the information bar. The text content is given in the corresponding functions in the Operator Interface Library and integrated into the test sequence. The text in the information bar is displayed until it is overwritten by a new text.

6.2.3.6 Statistic Display

Data is collated and displayed for the statistic display throughout the test sequence process.

Total	Displays the number of tests performed.
Pass Ratio	Displays the tests demarcated as PASS as a percentage.
since	Displays the date, from which the statistic data has been collated.

In order to ensure that the statistics can continue to be collated over a extended period of time and beyond, the corresponding data is stored in the configuration file when exiting the Generic Test Operator Interface R&S GTOP (see chapter 6.2.4). The next time the R&S GTOP is started this data is then loaded again and continued.

The <Configure><Statistic Options> menu item enables the dialog window for configuring the statistic options to be called up (see Figure 6-14).

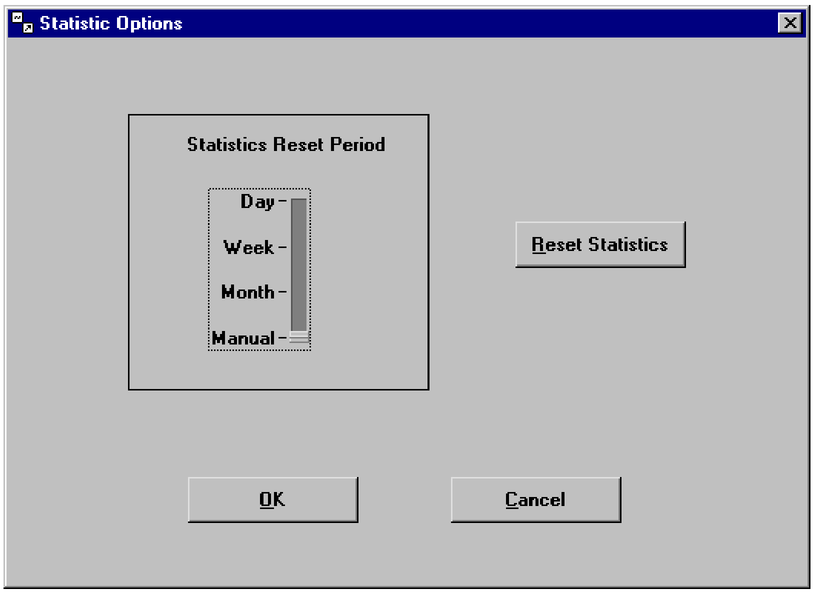


Figure 6-14 Statistic Options

Statistics Reset Period This selector switch enables the time period to be configured for determining when the statistic data should automatically be reset.

Day:	The statistic data is automatically reset at the end of a day.
Week:	The statistic data is automatically reset at the end of a week.
Month:	The statistic data is automatically reset at the end of a month.
Manual:	The statistic data can only be reset manually (Button **Reset Statistics**)

Reset Statistics Resets the statistic data.

OK The configurations made are saved in the configuration file and the dialog window is closed.

Cancel The configurations made are cancelled and the dialog window closed.

6.2.4 R&S GTOP Configuration File

The R&S GTOP configuration file is, as everywhere in the R&S GTOP, structured similar to a Windows INI file.

Example:

```
[OperatorInterface]
NumPanels = 2
Statistics_Type = "daily"

[Panel_1]
SequenceFile = "C:\Program Files\GTSL\Sequences\test.seq"
Statistics_Start = "2001-04-03"
Total_Ok = 24
Total_Tested = 48

[Panel_2]
SequenceFile = "C:\Program Files\GTSL\Sequences\test.seq"
Statistics_Start = "2001-04-03"
Total_Ok = 24
Total_Tested = 48
```

The configuration file contains a general section [OperatorInterface] and a channel-specific section [Panel_n].

[OperatorInterface]

`NumPanels =`	Shows the number of channels which are to be displayed in the Generic Test Operator Interface R&S GTOP. A channel-specific section [Panel_n] must be given for each channel. Permissible value: 1 or 2
`Statistics_Type =`	Indicates the period when the statistic data is to be automatically reset. Permissible data: – daily – weekly – monthly – manually

[Panel_n]

`SequenceFile =`	Indicates the path and file name of the test sequences to be performed in this channel.
`Statistics_Start =`	Indicates the date, from which the statistic data has been collated.
`Total_Ok =`	Indicates the number of tests concluded with a PASS.
`Total_Tested =`	Displays the number of tests performed.

The „NumPanels“ and „SequenceFile“ entries must be given in the configuration file. If these entries are missing, an error message is issued and the start of the Generic Test Operator Interface R&S GTOP is terminated.

All other entries relate to statistic data. This data is automatically entered into the configuration file using default values (see Table 6-2).

INI File Eintrag	Standardwert
`Statistics_Type`	„manually“
`Statistics_Start`	Today’s date in „yyyy-mm-dd“ format
`Total_Ok`	0
`Total_Tested`	0

Table 6-2 Configuration file Default Value

7 Test Libraries

The following sections briefly review the test functions which are available in the test libraries created by ROHDE & SCHWARZ.

NOTE:

A description of the individual test functions and their parameters will be found in the online help for the particular test library. The help files (.HLP) are in the directory `...\GTSL\BIN`.

7.1 Generic Test Libraries

7.1.1 Audio Analysis Library

7.1.1.1 General

Name of the dynamic link library (DLL):	AUDIOANL.DLL
Name of the help file (HLP):	AUDIOANL.HLP
License required:	R&S TS-LBAA
Supported devices:	Not required

The Audio Analysis Library offers functions to analyze audio waveform data in memory. The following analysis functions are supported:

- RMS calculation
- Single-/Multitone frequency response
- Distortion
- Filters (low-pass, high-pass, band-pass, band-stop, CCIR weighted/unweighted)
- Windows

7.1.1.2 Entries in PHYSICAL.INI

No entries required.

7.1.1.3 Entries in APPLICATION.INI

Section [bench->...]

Keyword	Value	Description
Trace	0 / 1	*Optional entry* Blocks the tracing function of the library (value = 0), enables the tracing function of the library (value = 1). Default = 0

7.1.1.4 Functions

Management	
Setup	AUDIOANL_Setup
Library Version	AUDIOANL_Lib_Version
Cleanup	AUDIOANL_Cleanup
Configuration	
Configure Filter	AUDIOANL_Configure_Filter
Calculation	
Frequency Response	AUDIOANL_Freq_Response
Distortion	AUDIOANL_Distortion
RMS	AUDIOANL_RMS
Filter Delay	AUDIOANL_Filter_Delay

7.1.2 DC Power Supply Test Library

7.1.2.1 General

Name of the dynamic link library (DLL):	DCPWR.DLL
Name of the help file (HLP):	DCPWR.HLP
License required	R&S TS-LBAS
Supported devices:	R&S TS-PSU Power Supply / Load Module R&S TS-PSU12 Power Supply / Load Module 12V Any DC Power Supply with an IVI device driver

The DCPWR Library controls one or more DC power supplies with drivers that conform to the IviDCPwr Class specification. It provides high level functions for controlling several DC power supply devices in one bench. Each power supply device provides one or more DC power channels.

7.1.2.2 Entries in PHYSICAL.INI

Section [device->...]

Keyword	Value	Description

Type	String	*Mandatory entry*
		for R&S TS-PSU : PSU
		for R&S TS-PSU12 : PSU12
		for other IviDcPwr compliant instruments : IVI_DCPWR

ResourceDesc	String	*Mandatory entry* VISA resource descriptor in the form PXI[segment number]::[device number]::[function]::INSTR CAN[board]::[controller]::[frame]::[slot] GPIB[board]::[primary address]::[secondary address]

DriverPrefix	String	*Mandatory entry*
		Prefix for the IVI driver functions
		for R&S TS-PSU: rspsu
		for R&S TS-PSU12: rspsu
		for others: driver dependent

Keyword	Value	Description

DriverDLL	String	*Mandatory entry*
		File name of the driver DLL
		for R&S TS-PSU: rspsu.dll
		for R&S TS-PSU12: rspsu.dll
		for others: driver dependent

DriverOption	String	*Optional entry* Option string being passed to the device driver during the driver's InitWithOptions function. See the online help file for the appropriate device driver.

7.1.2.3 Entries in APPLICATION.INI

Section [bench->...]

Keyword	Value	Description
DCPwrSupply<i>	String	*Mandatory entry* Reference to a DC power supply device section. <i> stands for a number 1,2,3,...,n. Numbers must be in ascending order without gaps. <i> may be omitted in the case it is 1.
DCPwrChannelTa-ble	String	*Mandatory entry* Reference to the application channel table section.
Simulation	0 / 1	*Optional entry* Enables/disables library-level simulation, default = 0
Trace	0 / 1	*Optional entry* Enables/disables tracing, default = 0

Section [io_channel->...] (Mandatory entries)

Contains a list of user-defined channel names with corresponding device names and device channel names. For details about channel name syntax see chapter 8.3.4.

Keyword	Value	Description
<user-defined name>	String	Physical channel description in the form <device name>!<device channel name>

7.1.2.4 Functions

Management	
Setup	DCPWR_Setup
Library Version	DCPWR_Lib_Version
Cleanup	DCPWR_Cleanup
Configuration	
Voltage Level	DCPWR_Conf_Voltage_Level
Overvoltage Protection	DCPWR_Conf_OVP
Current Limit	DCPWR_Conf_Current_Limit
Output Range	DCPWR_Conf_Output_Range
Output Enabled	DCPWR_Conf_Output_Enabled
Information	
Query Max Current Limit	DCPWR_Query_Max_Current_Limit
Query Max Voltage Level	DCPWR_Query_Max_Voltage_Level
Query Output State	DCPWR_Query_Output_State
Utility	
Reset	DCPWR_Reset
Reset Output Protection	DCPWR_Reset_Output_Protection
Measurement	
Measure	DCPWR_Measure
Trigger	
Configure Trigger Source	DCPWR_Conf_Trigger_Source
Configure Triggered Voltage Level	DCPWR_Conf_Triggered_Voltage_Level
Configure Triggered Current Limit	DCPWR_Conf_Triggered_Current_Limit
Initiate	DCPWR_Initiate
Abort	DCPWR_Abort
Send Software Trigger	DCPWR_Send_Software_Trigger
Instrument Driver Support	
Get Instrument Handle	DCPWR_Instrument_Get_Handle
TS-PSU Specific Functions	
Configure Mode	DCPWR_Conf_Mode
Configure Ground Relay	DCPWR_Conf_Ground_Relay
Configure Relay Protection	DCPWR_Conf_Relay_Protection
Configure Remote Sensing	DCPWR_Conf_Remote_Sensing
Configure PWM Output	DCPWR_Conf_Output_PWM
Configure Trigger Output	DCPWR_Conf_Trigger_Output
Configure Measurement	DCPWR_Conf_Measurement
Initiate Trigger	DCPWR_Initiate_Trigger
Wait Until Settled	DCPWR_Wait_Until_Settled
Acquisition	
Configure Acquisition	DCPWR_Conf_Acquisition
Initiate Acquisition	DCPWR_Initiate_Acquisition
Fetch Acquisition	DCPWR_Fetch_Acquisition
Abort Acquisition	DCPWR_Abort_Acquisition
Arbitrary Waveform Output	
Set Arbitrary Waveform	DCPWR_Set_Arb_Wfm
Configure Arbitrary Waveform	DCPWR_Conf_Arb_Wfm
Configure Arbitrary Waveform Abort	DCPWR_Conf_Arb_Wfm_Abort
Initiate Arbitrary Waveform	DCPWR_Initiate_Arb_Wfm
Abort Arbitrary Waveform	DCPWR_Abort_Arb_Wfm
Gated Output	

Configure Gated Output	DCPWR_Conf_Gated_Output
Enable Gated Output	DCPWR_Enable_Gated_Output
PAC Control	
Configure PAC Control	DCPWR_Conf_PAC_Control
Sink Modes	
Configure Constant Current	DCPWR_Conf_Const_Current
Configure Constant Resistance	DCPWR_Conf_Const_Resistance
Configure Constant Power	DCPWR_Conf_Const_Power
Query Sink State	DCPWR_Query_Sink_State

7.1.3 Digital I/O Manager Library

7.1.3.1 General

Name of the dynamic link library (DLL):	DIOMGR.DLL
Name of the help file (HLP):	DIOMGR.HLP
License required:	R&S TS-LBAS
Supported devices:	R&S TS-PDFT, Digital Functional Test Module R&S TS-PIO3B, Digital I/O Module

The Digital I/O Manager (DIO Manager) provides high level functions for digital functional tests based on one or more R&S TS-PDFT module(s). These functions include:

- Configuration of stimulus and response channels
- Application of binary stimulus patterns
- Collection of binary response data

Stimulus and response patterns can be executed at an arbitrary rate, usually under computer control. This is referred to as “Static Execution”. Application of stimulus and collection of response patterns at a fixed clock rate is referred to as “Dynamic Execution”.

The "Static Execution" functions of the DIO Manager not only support R&S TS-PDFT modules, but also R&S TS-PIO3B modules.

The DIO Manager supports static and dynamic pattern execution across several R&S TS-PDFT modules and also static pattern executions across several R&S TS-PIO3B modules. Stimulus and response channels can be defined through logical channel names.

Dynamic pattern sets can be designed graphically with the waveform editor of the ALTERA “Quartus II Web Edition” software. The DIO Manager imports the waveform file, executes the pattern set and exports the results into a file. Deviations from the expected patterns can be easily located by comparing both files in the Quartus waveform editor.

7.1.3.2 Entries in PHYSICAL.INI

Section [device->...]

Keyword	Value	Description
Type	String	*Mandatory entry* R&S TS-PDFT: pdft R&S TS-PIO3B: pio3b
ResourceDesc	String	*Mandatory entry* VISA resource descriptor in the form PXI[segment number]::[device number]::[function]::INSTR CAN[board]::[controller]::[frame]::[slot]
DriverPrefix	String	*Mandatory entry* prefix for the IVI driver functions, without underscore R&S TS-PDFT: rspdft R&S TS-PIO3B: rspio3b
DriverDLL	String	*Mandatory entry* File name of the driver DLL R&S TS-PDFT: rspdft.dll R&S TS-PIO3B: rspio3b.dll
DriverOption	String	*Optional entry* Option string being passed to the device driver during the Driver's InitWithOptions function. See the online help file for the appropriate device driver.

7.1.3.3 Entries in APPLICATION.INI

Section [bench->...]

Keyword	Value	Description
DIODevice<i>	String	*Mandatory entry* Refers to a section with digital I/O devices in PHYSICAL.INI <i> stands for a number from 1,2,3,...,40. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.
DIOChannelTable	String	*Mandatory entry* Refers to a section with defined channel names in APPLICATION.INI.

Keyword	Value	Description
DIOTriggerLine	String	*Optional entry* Specifies a PXI trigger line for synchronization of two or more R&S TS-PDFT modules Valid values : 0 - 7 Default : 1
Simulation	0 / 1	*Optional entry* Blocks simulation of all entered devices (value = 0). Enables simulation of the entered devices (value = 1). Default = 0
Trace	0 / 1	*Optional entry* Blocks the tracing function (value = 0), Enables the tracing function (value = 1). Default = 0
ChannelTableCase Sensitive	0 / 1	*Optional entry* The channel names in the channel table are treated case-sensitive (value = 1) or case-insensitive (value = 0).

Section [io_channel->...] (Optional entries)

Mandatory entry

Contains a list of user-specific channel names which are assigned to the physical device names and to the physical device channel names. The defined names apply only to the relevant application. For details about channel name syntax see chapter 8.3.4.

Keyword	Value	Description
<user-defined name>	String	Physical channel description in the form: <device name>!<device channel name>. Permissible R&S TS-PDFT channel names are: OUT1 - OUT32 (stimulus channesl) IN1 - IN32 (response channels) PO1 - PO4 (power output channels) Permissible R&S TS-PIO3B channel names are: PxIOy with x=0..7 and y=0..7(stimulus/response channels) P9TIOy with y=0..7 (stimulus/response channels) INH (inhibit stimulus channel)

7.1.3.4 Functions

Management	
Setup	DIOMGR_Setup
Library Version	DIOMGR_Lib_Version
Cleanup	DIOMGR_Cleanup
Configuration	
Configure Stimulus	DIOMGR_ConfigureStimulus
Configure Response	DIOMGR_ConfigureResponse
Configure Loopback	DIOMGR_ConfigureLoopback
Static Execution	
Port Stimulus	DIOMGR_PortStimulus
Port Response	DIOMGR_PortResponse
Dynamic Execution	
Configuration	
Load Waveform	DIOMGR_LoadWaveform
Configure Pattern Set Timing	DIOMGR_ConfigurePatternSetTiming
Save Waveform	DIOMGR_SaveWaveform
Unload Waveform	DIOMGR_UnloadWaveform
Action	
Execute Pattern Set	DIOMGR_ExecutePatternSet
Wait Until Pattern Set Complete	DIOMGR_WaitUntilPatternSetComplete
Abort Pattern Set	DIOMGR_AbortPatternSet
Results	
Number of Executed Patterns	DIOMGR_GetPatternSetExecutedPatternCount
Number of Failed Patterns	DIOMGR_GetPatternSetFailedPatternCount
Number of Failed Channels	DIOMGR_GetPatternSetFailedChannelCount
Failed Channel Names	DIOMGR_GetPatternSetFailedChannelNames
High/Low Data of a Channel	DIOMGR_GetPatternSetChannelData
Pass/Fail Results of a Channel	DIOMGR_GetPatternSetChannelResults
Instrument Driver Support	
Get Instrument Handle	DIOMGR_Instrument_Get_Handle

7.1.4 DMM Test Library

7.1.4.1 General

Name of the dynamic link library (DLL):	DMM.DLL
Name of the help file (HLP):	DMM.HLP
License required	R&S TS-LBAS
Supported devices:	National Instruments NI4060, any other DMM with IVI compliant driver R&S TS-PSAM

The DMM Library has been implemented to control numerous digital multimeter drivers that conform to the IviDmm Class specification. At present both the IVI-5 specification functions and those determined obsolete are supported.

The DMM Library provides high level functions for configuring and performing measurements.

7.1.4.2 Entries in PHYSICAL.INI

Section [device->...]

Keyword	Value	Description
Type	String	*Mandatory entry* NI4060 or IVI_DMM psam = R&S TS-PSAM Analog Source and Measurement Module
ResourceDesc	String	*Mandatory entry* VISA resource descriptor in the form DAQ::[deviceNumber]::INSTR (only NI460) PXI[segment number]::[device number]::[function]::INSTR
DriverPrefix	String	*Mandatory entry* Prefix for the IVI driver functions, without underscore: NI4060: niDMM IVI_DMM: driver dependent R&S TS-PSAM: rspsam

Keyword	Value	Description
DriverDLL	String	*Mandatory entry* File name of the driver DLL NI4060: nidmm_32.dll IVI_DMM: driver dependent R&S TS-PSAM: rspsam.dll
DriverOption	String	*Optional entry* Option string being passed to the device driver during the Driver_Init function. See the online help file for the appropriate device driver.
PowerLineFrequen-cy	50 or 60	*Optional entry* Sets the power line frequency in Hz, default = 60 (not for R&S TS-PSAM)

7.1.4.3 Entries in APPLICATION.INI

Section [bench->...]

Keyword	Value	Description
DigitalMultimeter	String	*Mandatory entry* Reference to multimeter device entry
Simulation	0 / 1	*Optional entry* Enables/disables library-level simulation, default = 0
Trace	0 / 1	*Optional entry* Enables/disables tracing, default = 0

7.1.4.4 Functions

```
Setup                                 DMM_Setup
Library Version                       DMM_Lib_Version
Configuration Functions
   Configure Measurement              DMM_Conf_Measurement
   Configure Trigger                  DMM_Conf_Trigger
   Configure Trigger Slope            DMM_Conf_Trigger_Slope
   Configure Auto Zero Mode           DMM_Conf_Auto_Zero_Mode
   Configure AC Bandwidth             DMM_Conf_AC_Bandwidth
   Configure Power Line Frequency     DMM_Conf_Power_Line_Frequency
   Configure Measurement Complete     DMM_Conf_Meas_Complete_Dest
   Configure Multi Point              DMM_Conf_Multi_Point
Measurement Functions
   Measure DC Voltage                 DMM_Meas_DC_Voltage
   Measure DC Current                 DMM_Meas_DC_Current
```

Measure AC Voltage	DMM_Meas_AC_Voltage
Measure AC Current	DMM_Meas_AC_Current
Measure Resistance	DMM_Meas_Resistance
Low Level Measurement Functions	
Initiate	DMM_Initiate
Fetch	DMM_Fetch
Abort	DMM_Abort
Send Software Trigger	DMM_Send_Software_Trigger
Utility Functions	
Reset	DMM_Reset
TS-PSAM Specific Functions	
Configure Lowpass Filter	DMM_Conf_Lowpass_Filter
Configure Trigger Signal	DMM_Conf_Trigger_Signal
Configure Analog Trigger	DMM_Conf_Analog_Trigger
Configure Trigger Output	DMM_Conf_Trigger_Output
Enable Trigger Output	DMM_Enable_Trigger_Output
Send Software Signal	DMM_Send_Software_Signal
Configure Coupling Relays	DMM_Conf_Coupling_Relays
Configure Ground Relay	DMM_Conf_Ground_Relay
Cleanup	DMM_Cleanup

7.1.5 Factory Toolbox Library

7.1.5.1 General

Name of the dynamic link library (DLL):	FTBLIB.DLL
Name of the help file (HLP):	FTBLIB.HLP
License required	R&S TS-LBAS
Supported devices:	Digital IO Module 3B R&S TS-PIO3B

The "Factory Toolbox" library offers functions for identifying adapters via the Digital IO Module 3B R&S TS-PIO3B. Two methods are available for this purpose:

- Parallel adapter identification via ports
- Serial adapter identification via SPI-EEPROM

The identification of test adapters is parameterized either via entries in a configuration file for the application layer (APPLICATION.INI) or via function calls in the test program. An R&S TS-PIO3B module must be entered in the file for configuring the system (PHYSICAL.INI).

Parallel Adapter Identification via Ports

The parallel adapter identification is especially easy to implement but requires one or two complete 8 bit IO ports. For this purpose, the 8 open drain ports (P0 to P7) are available on the R&S TS-PIO3B module. To be able to identify an adapter uniquely, it is only necessary to connect wires in the adapter with GND. Port bits that have not been wired are read as ‚high' by the internal pull-up resistors.

Serial Adapter Identification via SPI-EEPROM

The serial adapter identification uses an SPI-EEPROM in the adapter and can be set up easily in connection with an R&S TS-PTRF. The Atmel AT25160 module, a 16 kBit (2 kByte) EEPROM for SPI is supported. In this way, all the open drain IO ports remain free and only one SPI Chip-Select signal (E_CSx) of the R&S TS-PTRF is occupied. In the following example, E_CS3 (port 3) is used for the adapter identification. In this case you have to observe that the SPI signals (E_MOSI, E_MISO and E_SCLK) as well as the selected Chip-Select signal must be connected to the front panel (X10) via jumpers on the R&S TS-PTRF module.

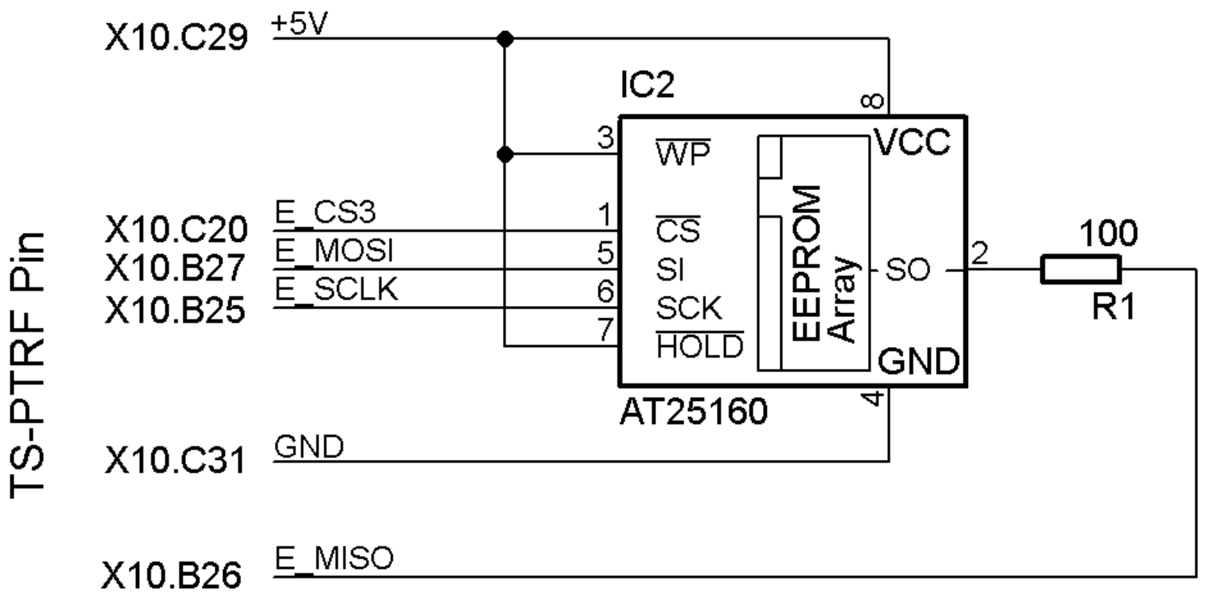


Figure 7-1 Serial adapter identification via SPI-EEPROM

7.1.5.2 Entries in PHYSICAL.INI

An R&S TS-PIO3B module must be installed in the system and the related entry must be available in the configuration file for the system.

Key	Value	Description
Type	String	*Mandatory entry* PIO3B
ResourceDesc	String	*Mandatory entry* resource descriptor in the form CAN[board]::[controller]::[frame]::[slot]
DriverPrefix	String	*Mandatory entry* Prefix for the IVI driver functions, without underscore: rspio3b
DriverDll	String	*Mandatory entry* File name of the driver DLL rspio3b.dll
DriverOption	String	*Optional Entry* Option string being passed to the device driver during the Driver_Init function. See the online help file for the appropriate device driver.

Key	Value	Description
Description	String	*OptionalEntry* Gives a detailed description of the defined device..

7.1.5.3 Entries in APPLICATION.INI

This configuration file is usually created individually for each unit under test / each test program. It can be assigned any name and stored in any directory. The following table provides an overview of the keywords.

Section [bench->...]

Key	Value	Description
AdaIdentDevice	String	*Optional entry* Refers to the device section of the TS-PIO3B (e. g. AdaIdentDevice = device->rspio3b_1) if the adapter identification functionality.
AdaIdentParPortLo	0…7	*Optional entry* R&S TS-PIO3B port from which the lower 8 bits of the adapter identification are read. Default = 0
AdaIdentParPortHi	0…7	*Optional entry* R&S TS-PIO3B port from which the higher 8 bits of the adapter identification are read. If only one 8-bit identification is to be used, the same port as in the "AdaIdentPortLo" parameter is specified here. The higher 8 bits in the result will then be set to 0. Default = 0 (8 bit identification via port 0)
AdaIdentSpiPort	0..7	*Optional entry* Controls the Chip-Select generation via an R&S TS-PTRF module for the adapter identification with R&S TS-PIO3B via an SPI EEPROM. If the R&S TS-PTRF module is not available, this parameter will be ignored. Default = 0
Simulation	0 / 1	*Optional entry* Blocks the simulation of the entered devices (value = 0). Enables simulation of the entered devices (value = 1). Default = 0

Key	Value	Description
Trace	0 / 1	Optional entry Blocks the tracing function of the library (value = 0). Enables the tracing function of the library (value = 1). Default = 0

7.1.5.4 Functions

The following routines are available for identifying a test adapter. Refer to the help file (FTBLIB.HLP) for a description.

Management

Setup	FTBLIB_Setup
Library Version	FTBLIB_Lib_Version
Cleanup	FTBLIB_Cleanup

Adapter Identification

Parallel Port

Parallel Config Port	FTBLIB_AdaIdentParConfigPort
Parallel Read	FTBLIB_AdaIdentParRead

SPI EEPROM

SPI Config Port	FTBLIB_AdaIdentSpiConfigPort
SPI Read	FTBLIB_AdaIdentSpiRead
SPI Write	FTBLIB_AdaIdentSpiWrite

7.1.6 Function Generator Library

7.1.6.1 General

Name of the dynamic link library (DLL):	FUNCGEN.DLL
Name of the help file (HLP):	FUNCGEN.HLP
License required	R&S TS-LBAS
Supported devices:	R&S TS-PFG Arbitrary Waveform Generator Any other waveform generator with IVI compliant driver.

The Function Generator Library provides functions for waveform generators:

- Standard Waveforms
- Arbitrary Waveforms
- Arbitrary Waveform Sequences

7.1.6.2 Entries in PHYSICAL.INI

Section [device->...]

Keyword	Value	Description

Type	String	*Mandatory entry*
		PFG = R&S TS-PFG Arbitrary Waveform Generator
		IVI_FGEN = other IVI compliant generator

ResourceDesc	String	*Mandatory entry* VISA resource descriptor in the form PXI[segment number]::[device number]::[function]::INSTR
DriverPrefix	String	*Mandatory entry* Prefix for the IVI driver functions, without underscore: IVI_FGEN: driver dependent R&S TS-PFG: rspfg
DriverDll	String	*Mandatory entry* File name of the driver DLL IVI_FGEN: driver dependent R&S TS-PFG: rspfg.dll

Keyword	Value	Description
DriverOption	String	*Optional entry* Option string being passed to the device driver during the Driver_Init function. See the online help file for the appropriate device driver.

7.1.6.3 Entries in APPLICATION.INI

Section [bench->...]

Keyword	Value	Description
FunctionGenerator	String	*Mandatory entry* Reference to the device entry of the function generator in PHYSICAL.INI.
Simulation	0 / 1	*Optional entry* Blocks the simulation of the entered devices (value = 0). Enables simulation of the entered devices (value = 1). Default = 0
Trace	0 / 1	*Optional entry* Blocks the tracing function of the library (value = 0), enables the tracing function of the library (value = 1). Default = 0

7.1.6.4 Functions

Setup	FUNCGEN_Setup
Library Version	FUNCGEN_Lib_Version
Basic Instrument Operation	
Configure Operation Mode	FUNCGEN_ConfigureOperationMode
Configure Output Mode	FUNCGEN_ConfigureOutputMode
Configure Ref Clock Source	FUNCGEN_ConfigureRefClockSource
Configure Output Impedance	FUNCGEN_ConfigureOutputImpedance
Configure Output Enabled	FUNCGEN_ConfigureOutputEnabled
Initiate Generation	FUNCGEN_InitiateGeneration
Abort Generation	FUNCGEN_AbortGeneration
Standard Function Output	
Configure Standard Waveform	FUNCGEN_ConfigureStandardWaveform
Arbitrary Waveform Output	
Query Arb Waveform Capabilities	FUNCGEN_QueryArbWfmCapabilities
Create Arbitrary Waveform	FUNCGEN_CreateArbWaveform
Configure Arbitrary Waveform	FUNCGEN_ConfigureArbWaveform
Configure Sample Rate	FUNCGEN_ConfigureSampleRate
Clear Arbitrary Waveform	FUNCGEN_ClearArbWaveform
Arbitrary Frequency	
Configure Arbitrary Frequency	FUNCGEN_ConfigureArbFrequency
Arbitrary Sequence Output	
Query Arb Sequence Capabilities	FUNCGEN_QueryArbSeqCapabilities
Create Arbitrary Sequence	FUNCGEN_CreateArbSequence
Configure Arbitrary Sequence	FUNCGEN_ConfigureArbSequence
Clear Arbitrary Sequence	FUNCGEN_ClearArbSequence
Clear Arbitrary Memory	FUNCGEN_ClearArbMemory
Triggering	
Configure Trigger Source	FUNCGEN_ConfigureTriggerSource
Software Triggering	
Send Software Trigger	FUNCGEN_SendSoftwareTrigger
Burst Configuration	
Configure Burst Count	FUNCGEN_ConfigureBurstCount
Utility Functions	
Import Waveform Data	FUNCGEN_ImportWaveformData
Reset	FUNCGEN_Reset
TS-PFG Specific Functions	
Configure Trigger Delay	FUNCGEN_ConfigureTriggerDelay
Configure Filter	FUNCGEN_ConfigureFilter
Configure Arbitrary Marker	FUNCGEN_ConfigureArbMarker
Configure Marker Output	FUNCGEN_ConfigureMarkerOutput
Configure Coupling Relays	FUNCGEN_ConfigureCouplingRelays
Configure Ground Relay	FUNCGEN_ConfigureGroundRelay
Cleanup	FUNCGEN_Cleanup

7.1.7 Operator Interface Library

7.1.7.1 General

Name of the dynamic link library (DLL):	OPERINT.DLL
Name of the help file (HLP):	OPERINT.HLP
License required	R&S TS-LBAS
Supported devices:	not usable

The Operator Interface Library offers functions for interaction between the TestStand sequence and the R&S GTSL Operator Interface:

- Display of Banners and other Information
- Dialog Boxes
- User Input

If the R&S GTSL Operator Interface is not available (e.g. when a sequence is started from the TestStand Sequence Editor), the dialog boxes are replaced by simple pop-up dialogs.

This library requires TestStand as test executive, it cannot be run in other environments.

7.1.7.2 Entries in PHYSICAL.INI

No entries

7.1.7.3 Entries in APPLICATION.INI

Section [bench->...]

Keyword	Value	Description
Trace	0 / 1	*Optional entry* 0 : Disable tracing 1 : Enable tracing Default: 0

7.1.7.4 Functions

Setup	OPERINT_Setup
Library Version	OPERINT_Lib_Version
Information	
Get Associated Channel Number	OPERINT_Get_Channel
Display Functions	
Show Banner	OPERINT_Show_Banner
Show Customized Banner	OPERINT_Show_Custom_Banner
Hide Banner	OPERINT_Hide_Banner
Customized Dialog	OPERINT_Custom_Dialog
Customized Input Prompt	OPERINT_Custom_Prompt
Display Info Line	OPERINT_Display_Info
Cleanup	OPERINT_Cleanup

7.1.8 QUOTIS Logging Library

7.1.8.1 General

Name of the dynamic link library (DLL):	QDILOG.DLL
Name of the help file (HLP):	QDILOG.HLP
License required	R&S TS-LBAS
Supported devices:	not usable

The QUOTIS Logging Interface Library offers functions for logging test results. Test reports are stored as XML files. These files can be imported into the QUOTIS® software for quality analysis and paperless repair.

You will find more information about QUOTIS on the R&S GTSL CD in the folder "`Runtime Setup\7 - QUOTIS`" or visit the manufacturers website www.sekas.de.

QUOTIS® is a trademark of SEKAS GmbH.

7.1.8.2 Entries in PHYSICAL.INI

No entries

7.1.8.3 Entries in APPLICATION.INI

No entries

7.1.8.4 Functions

Setup	QDILOG_Setup
Library Version	QDILOG_Lib_Version
Logging Test Results	
Create new Log File	QDILOG_Log_Create
Open Sub-Test	QDILOG_Open_Subtest
Close Sub-Test	QDILOG_Close_Subtest
Add Step	QDILOG_Add_Step
Add Step Details	
Component Reference	QDILOG_Add_Detail_Component_Reference
Measurement Results	QDILOG_Add_Detail_MeasResults
Measurement Results String	QDILOG_Add_Detail_MeasResults_String
Measurement Results Comparison String	QDILOG_Add_Detail_MeasResults_ComparisonString
File	QDILOG_Add_Detail_File
Node List	QDILOG_Add_Detail_NodeList
Close Log File	QDILOG_Log_Close
Cancel Logging	QDILOG_Log_Cancel
Custom Configuration	
Set Report File	QDILOG_Config_Set_Reportfile
Get Report File Information	QDILOG_Config_Get_ReportFile_Info
Cleanup	QDILOG_Cleanup

7.1.9 Resource Manager Library

7.1.9.1 General

Name of the dynamic link library (DLL):	RESMGR.DLL
Name of the help file (HLP):	RESMGR.HLP
License required	R&S TS-LBAS
Supported devices:	not usable

The Resource Manager Library provides functions for managing the hardware used in the test system.

7.1.9.2 Entries in PHYSICAL.INI

No entries

7.1.9.3 Entries in APPLICATION.INI

Section [Resource Manager]

Optional entry

Controls the tracing properties of the Resource Manager.

Keyword	Value	Description
Trace	0 / 1	Blocks the tracing function (value = 0), enables the tracing function (value = 1). Default = 0
TraceFile	String	Defines the path and the name of the trace file. Default = „“
TraceToScreen	0 / 1	The tracing information is displayed on the standard screen (value = 1). Default = 0
TraceTimeStamp	0 / 1	Writes the time of day at the start of each tracing line (value = 1). Default = 0
TraceThreadID	0 / 1	Writes the ID of the current thread at the start of each tracing line (value = 1). Default = 0
TraceAppend	0 / 1	Appends lines to existing trace file (value = 1) Overwrites trace file (value = 0, default)

Keyword	Value	Description
TraceAutoFlush	0 / 1	Writes trace lines immediately to disk (value =1) Buffers disk write operations (value = 0, default) Note that enabling this feature may degrade application performance significantly.

7.1.9.4 Functions

Management	
Setup	RESMGR_Setup
Library Version	RESMGR_Lib_Version
Cleanup	RESMGR_Cleanup
Resource Functions	
Allocate Resource	RESMGR_Alloc_Resource
Free Resource	RESMGR_Free_Resource
Informational	
Resource Type	RESMGR_Get_Resource_Type
Resource Name	RESMGR_Get_Resource_Name
Get Value	RESMGR_Get_Value
Compare Value	RESMGR_Compare_Value
Number of Sections	RESMGR_Number_Of_Sections
Nth Section Name	RESMGR_Nth_Section_Name
Number of Keys	RESMGR_Number_Of_Keys
Nth Key Name	RESMGR_Nth_Key_Name
Key Value	RESMGR_Get_Key_Value
Device Sessions	
Open Session	RESMGR_Open_Session
Get Session Handle	RESMGR_Get_Session_Handle
Set Session Handle	RESMGR_Set_Session_Handle
Close Session	RESMGR_Close_Session
Open Sub-Session	RESMGR_Open_SubSession
Get Session Sub-Handle	RESMGR_Get_Session_SubHandle
Set Session Sub-Handle	RESMGR_Set_Session_SubHandle
Close Sub-Session	RESMGR_Close_SubSession
Dynamic memory Management	
Allocate Memory	RESMGR_Alloc_Memory
Get Memory Pointer	RESMGR_Get_Mem_Ptr
Free Memory	RESMGR_Free_Memory
Allocate Shared Memory	RESMGR_Alloc_Shared_Memory
Lock Shared Memory	RESMGR_Lock_Shared_Memory
Unlock Shared Memory	RESMGR_Unlock_Shared_Memory
Free Shared Memory	RESMGR_Free_Shared_Memory
Locking	
Lock Device	RESMGR_Lock_Device
Unlock Device	RESMGR_Unlock_Device
Support Functions	
Read System Identification	RESMGR_Read_ROM
Enable Tracing	RESMGR_Enable_Tracing
Trace	RESMGR_Trace
Set Trace Flag	RESMGR_Set_Trace_Flag
Get Trace Flag	RESMGR_Get_Trace_Flag

7.1.10 Self Test Support Library

7.1.10.1 General

Name of the dynamic link library (DLL):	SFT.DLL
Name of the help file (HLP):	SFT.HLP
License required	R&S TS-LBAS
Supported devices	Self Test Multimeter: National Instruments NI4060 with Self Test Matrix Card R&S TS-PMA or Self Test Multimeter with integrated matrix R&S TS-PSAM

The self test support library contains functions which are common for all self test modules like measurements, report generation, run-time state handling and operator dialog functions.

7.1.10.2 Entries in PHYSICAL.INI

Section [device->...]

Keyword	Value	Description
SFTDLL	String	*Mandatory entry* File name of the self test DLL R&S TS-PAM : sftmpam.dll R&S TS-PDFT : sftmpdft.dll R&S TS-PFG : sftmpft.dll R&S TS-PICT : sftmpict.dll R&S TS-PIO2: sftmpio2.dll R&S TS-PMA : sftmpma.dll R&S TS-PMB : sftmpmb.dll R&S TS-PRL1 : sftmprl1.dll R&S TS-PSAM : sftmpsam.dll R&S TS-PSM1 : sftmpsm1.dll R&S TS-PSM2 : sftmpsm2.dll R&S TS-PSU : sftmpsu.dll R&S TS-PSU12 : sftmpsu.dll R&S TS-PSYS : sftmpsys.dll

Keyword	Value	Description
SFTPrefix	String	*Mandatory entry* Prefix for the self test functions, without underscore. This entry is case sensitive: R&S TS-PAM : SFTMPAM R&S TS-PDFT : SFTMPDFT R&S TS-PFG : SFTMPFT R&S TS-PICT : SFTMPICT R&S TS-PIO2: SFTMPIO2 R&S TS-PMA : SFTPMA R&S TS-PMB : SFTMPMB R&S TS-PRL1 : SFTMPRL1 R&S TS-PSAM : SFTMPSAM R&S TS-PSM1 : SFTMPSM1 R&S TS-PSM2 : SFTMPSM2 R&S TS-PSU : SFTMPSU R&S TS-PSU12 : SFTMPSU R&S TS-PSYS : SFTMPSYS

NOTE:

These entries in the physical ini file inform the TSVP self test frame which devices can be tested during the self test. If both keys are present, the device can be tested during self test.

Example:

```
[device->RelayCard1]
Description = TS-PRL1 in Slot 7
Type = PRL1
ResourceDesc = PXI0::1::17
DriverDll = rsprl1.dll
DriverPrefix = rsprl1
SFTDll = SFTMPRL1.DLL
SFTPrefix = SFTMPRL1
```

Example of a physical layer .ini file for a R&S CompactTSVP system:

If the recommended self test hardware is installed, the following device section must be added:

```
[device->psam]
Description  = "TS-PSAM, source and measurement module, Slot 3"
Type         = PSAM
```

```
ResourceDesc = PXI1::15::0::INSTR
DriverDll    = rspsam.dll
DriverPrefix = rspsam
DriverOption = "Simulate=0,RangeCheck=1"
; Note: the self test DLL and prefix keywords must be removed for the
;       first TS-PSAM module, because it is already tested in the
;       basic self test.
;SFTDll       = sftmpsam.dll
;SFTPrefix    = SFTMPSAM
```

NOTE:

- **The type of the digital multimeter and the self test switch device must be "PSAM".**
- **The "ResourceDesc" keys must match your hardware configuration.**
- **Because this device is the self test instrumentation and under control of the self test library, this sections must not have the keys "SFTDll" and "SFTPrefix".**

In order to test a Rohde & Schwarz CompactTSVP module (R&S TS-PFG, R&S TS-PMB, ...) the device section of these modules must contain the entries "SFTDll and SFTPrefix". The following sections show the entries for a R&S TS-PFG and a R&S TS-PMB module:

```
[device->pfg]
Description  = "TS-PFG, arbitrary function generator module, Slot 4"
Type         = PFG
ResourceDesc = PXI1::14::0::INSTR
DriverDll    = rspfg.dll
DriverPrefix = rspfg
SFTDll       = sftmpfg.dll
SFTPrefix    = SFTMPFG

[device->pmb]
Description  = "TS-PMB, matrix module, Slot 10"
Type         = PMB
ResourceDesc = CAN0::0::1::10
DriverDll    = rspmb.dll
DriverPrefix = rspmb
SFTDll       = sftmpmb.dll
SFTPrefix    = SFTMPMB
```

NOTE:

The values of the keys "SFTPrefix" and "DriverPrefix" are case sensitive

The files "`CompactTSVP_physical.ini`" and "`SFT_CompactTSVP_application.ini`" in the `GTSL\Configuration` subdirectory are an example for the self test configuration of a R&S CompactTSVP test system.

Example of a physical layer .ini file for a TSVP system:

If the recommended self test hardware is installed, the following two device sections must be added:

```
[device->SftRelayCard]
Description = "Self Test Matrix Card"
Type = PMA1
ResourceDesc = PXI2::11::0::INSTR
DriverPrefix = rspma
DriverDll = rspma.dll
DriverOption = "Simulate=0,DriverSetup=MCR:FFFFFFF6 CRAuto:1 BusSel:0"

[device->SftDMM]
Description = "Self Test Digital Multimeter"
type = NI4060
ResourceDesc = DAQ::2::INSTR
DriverPrefix = niDMM
DriverDll = nidmm_32.dll
DriverOption = "Simulate=0,DriverSetup=PXI-4060"
PowerLineFrequency = 50
```

NOTE:

- **The type of the switch device must be "PMA1"**
- **The DriverSetup attribute in the DriverOption string of the PMA1 card must have the parameter setting "CRAuto:1"**
- **The type of the digital multimeter must be "NI4060"**
- **The "ResourceDesc" keys must match your hardware configuration**
- **Because these two devices are the self test instrumentation and under control of the self test library these sections must not have the keys "SFTDll" and "SFTPrefix".**

In order to test a Rohde & Schwarz TSVP module (R&S TS-PRL1, R&S TS-PMA) the device section of these cards must contain the new entries "SFTDll and SFTPrefix". The following sections show the entries for a R&S TS-PRL1 and a R&S TS-PMA card:

```
[device->RelayCard1]
Description = "Relay Card 1"
Type = PRL1
ResourceDesc = PXI3::10::0::INSTR
DriverPrefix = rsprl1
DriverDll = rsprl1.dll
DriverOption = "Simulate=0"
SFTDll = SFTMPRL1.DLL
SFTPrefix = SFTMPRL1

[device->MatrixCard1]
Description = "Matrix Card 1"
Type = PMA1
ResourceDesc = PXI1::11::0::INSTR
DriverPrefix = rspma
DriverDll = rspma.dll
DriverOption = "Simulate=0,DriverSetup=MCR:FFFFFFF6 CRAuto:0 BusSel:0"
SFTDll = SFTMPMA.DLL
SFTPrefix = SFTMPMA
```

NOTE:

- **The values of the keys "SFTPrefix" and "DriverPrefix" are case sensitive**
- **The DriverSetup attribute in the DriverOption string of a PMA card must have the parameter setting "CRAuto:0" for self test! This can also be configured in the self test application .ini file.**

The files "`demo_physical.ini`" and "`sft_7100_application.ini`" in the `GTSL\Configuration` subdirectory are an example for the self test configuration of a single channel TS7100 test system.

7.1.10.3 Entries in APPLICATION.INI

Section [bench->SFT]

Keyword	Value	Description
DigitalMultimeter	String	*Mandatory entry* Bench device link to the device entry of the multimeter: For R&S TSVP: National Instruments DMM 4060 For R&S CompactTSVP: R&S TS-PSAM
SwitchDevice	String	*Mandatory entry* Bench device link to the device entry of the matrix for the multimeter (R&S TS-PMA1)
Trace	0 / 1	*Optional entry* Enables/disables tracing, default = 0

Section [SftOptions]

Keyword	Value	Description
SystemName	String	The Name of the system to be tested. This Name appears in the header of the self test report. Default “-”
SFTFixture	0 / 1	1 : TSVP self test fixture is connected to the modules 0 : no SFT fixture present Default 1
ManualInterventions	0 / 1	1 : Additional tests which require user interaction 0 : SFT runs without further interaction. Default 1
ReportFile	String	Path and filename of report file Default "C:\TEMP\SFT_Report.txt"
ReportStyle	1 / 2 / 3 /...	1 : Only errors are reported 2 : Only a short report is generated 3 : Generates a full report all other entries: A full report is generated Default 1
ReportAppend	0 / 1	1 : Report is appended to existing file 0 : Existing report file is overwritten Default 0

Keyword	Value	Description
SuppressDialog	0 / 1	1 : Options dialog is not displayed 0 : Options dialog is displayed Default 0
StopOnFirstFailure	0 / 1	1 : SFT is aborted on first test case failure 0 : SFT is not aborted on first test case failure Default 0

Section [SftParts]

PartX	<PartName>, <BenchName>,<SelectFlag>

The keys have the format "PartX". Where X ist a continuous counter starting at 1. The values are comma separated lists with the following entries:

Part Name	An arbitrary name for a part to test. All parts must have a unique name.
Bench Name	Name of the corresponding bench
Select Flag	Default value for the selection 1: Part selected 0: Part not selcted

7.1.10.4 Functions

```
Management
     Setup                              SFT_Setup
     Library Version                    SFT_Lib_Version
     Cleanup                            SFT_Cleanup
  Measurement
     Dmm_reset                          SFT_Dmm_reset
     Dmm_Connect                        SFT_Dmm_Connect
     Dmm_Disconnect                     SFT_Dmm_Disconnect
     Dmm_DisconnectAll                  SFT_Dmm_DisconnectAll
     Dmm_MeasDelay                      SFT_Dmm_MeasDelay
     Dmm_WaitForDebounce                SFT_Dmm_WaitForDebounce
     Dmm_ConfigureMeasurement           SFT_Dmm_ConfigureMeasurement
     Dmm_ConfigureAutoZeroMode          SFT_Dmm_ConfigureAutoZeroMode
     Dmm_Read                           SFT_Dmm_Read
     Dmm_AverageMeasurement             SFT_Dmm_AverageMeasurement
     Dmm_checkForExternVoltage          SFT_Dmm_checkForExternVoltage
     Dmm_MeasureContact                 SFT_Dmm_MeasureContact
     Dmm_MeasureIsolation               SFT_Dmm_MeasureIsolation
  Trigger
```

Dmm_TriggerConfSignal	SFT_Dmm_TriggerConfSignal
Dmm_ConfigureTrigger	SFT_Dmm_ConfigureTrigger
Dmm_EnableTriggerline	SFT_Dmm_EnableTriggerline
Dmm_Initiate	SFT_Dmm_Initiate
Dmm_trig_SendSoftwareSignal	SFT_Dmm_trig_SendSoftwareSignal
Dmm_Fetch	SFT_Dmm_Fetch
DC_Source	
dcs_ConfigureVoltageLevel	SFT_dcs_ConfigureVoltageLevel
dcs_ConfigureCurrentLimitRange	SFT_dcs_ConfigureCurrentLimitRange
dcs_ConfigureCurrentLimit	SFT_dcs_ConfigureCurrentLimit
dcs_ConfigureOutputEnabled	SFT_dcs_ConfigureOutputEnabled
dcs_QueryOutputState	SFT_dcs_QueryOutputState
CNX	
cnx_ConfigureSwitches	SFT_cnx_ConfigureSwitches
cnx_Gnd	SFT_cnx_Gnd
cnx_DmmToGnd	SFT_cnx_DmmToGnd
cnx_Matrix	SFT_cnx_Matrix
cnx_Coupling	SFT_cnx_Coupling
Report	
CommentAddItem	SFT_CommentAddItem
WarningAddItem	SFT_WarningAddItem
ErrorAddItem	SFT_ErrorAddItem
ResultTextAddItem	SFT_ResultTextAddItem
ProgramErrorAddItem	SFT_ProgramErrorAddItem
TableAddItem	SFT_TableAddItem
TableColumnSetAttrInt	SFT_TableColumnSetAttrInt
TableColumnSetAttrString	SFT_TableColumnSetAttrString
TableCellSetValueInt	SFT_TableCellSetValueInt
TableCellSetValueDouble	SFT_TableCellSetValueDouble
TableCellSetValueString	SFT_TableCellSetValueString
ResultTabAddItem	SFT_ResultTabAddItem
ResultTabLineGetAttrInt	SFT_ResultTabLineGetAttrInt
ResultTabLineSetAttrDouble	SFT_ResultTabLineSetAttrDouble
ResultTabLineSetAttrString	SFT_ResultTabLineSetAttrString
ResultTabSetAttrInt	SFT_ResultTabSetAttrInt
ResultTabColumnSetAttrInt	SFT_ResultTabColumnSetAttrInt
ResultTabLineCalcAttrStatus	SFT_ResultTabLineCalcAttrStatus
Run-time State	
OptionGetAttrInt	SFT_OptionGetAttrInt
PartSelect	SFT_PartSelect
PartGetAttrInt	SFT_PartGetAttrInt
PartGetAttrString	SFT_PartGetAttrString
PartSetAttrInt	SFT_PartSetAttrInt
ComponentAddItem	SFT_ComponentAddItem
ComponentSelect	SFT_ComponentSelect
ComponentSelectByIndex	SFT_ComponentSelectByIndex
ComponentSetAttrInt	SFT_ComponentSetAttrInt
ComponentGetAttrInt	SFT_ComponentGetAttrInt
ComponentSetAttrString	SFT_ComponentSetAttrString
ComponentGetAttrString	SFT_ComponentGetAttrString
TestCaseAddItem	SFT_TestCaseAddItem

`TestCaseSelect`	`SFT_TestCaseSelect`
`TestCaseSelectByIndex`	`SFT_TestCaseSelectByIndex`
`TestCaseSetAttrInt`	`SFT_TestCaseSetAttrInt`
`TestCaseGetAttrInt`	`SFT_TestCaseGetAttrInt`
`TestCaseSetAttrString`	`SFT_TestCaseSetAttrString`
`Dialog`	
`ComponentShowDialog`	`SFT_ComponentShowDialog`
`DlgProgressPrintInfo`	`SFT_DlgProgressPrintInfo`
`Utility`	
`Dmm_Test`	`SFT_Dmm_Test`
`Dmm_writeRegister`	`SFT_Dmm_writeRegister`
`Dmm_readRegister`	`SFT_Dmm_readRegister`

7.1.11 Signal Analyzer Library

7.1.11.1 General

Name of the dynamic link library (DLL):	SIGANL.DLL
Name of the help file (HLP):	SIGANL.HLP
License required	R&S TS-LBAS
Supported devices:	R&S TS-PAM, Analyzer Module Any other module with IVI Scope compliant driv-er.

The Signal Analyzer Library provides configuration and measurement functions for waveform analyzers / digital oscilloscopes compliant with the IVI-4.1 IviScope instrument class. Furthermore it offers generic waveform analysis and utility functions.

These functions include:

- Channel configuration
- Time base configuration
- Trigger configuration
- Waveform acquisition

- Waveform parameter measurement (Average, RMS)
- Frequency measurement
- Event counting (Slope, Peak)
- Time measurement between events
- Waveform comparison
- Calculation of reference waveforms
- Waveform import/export as file
- Waveform display

7.1.11.2 Entries in PHYSICAL.INI

Section [device->...]

Keyword	Value	Description

Type	String	*Mandatory entry*
		for R&S TS-PAM: PAM
		for other IviScope compliant instruments: IVI_SCOPE

ResourceDesc	String	*Mandatory entry* VISA resource descriptor in the form PXI[segment number]::[device number]::[function]::IN-STR

DriverPrefix	String	*Mandatory entry*
		prefix for the IVI driver functions, without underscore
		for R&S TS-PAM: rspam
		for others: driver dependent

Driver DLL	String	*Mandatory entry*
		File name of the driver DLL
		for R&S TS-PAM: rspam.dll
		for others: driver dependent

DriverOption	String	*Optional entry* Option string being passed to the device driver during the Driver's InitWithOptions function. See the online help file for the appropriate device driver.

7.1.11.3 Entries in APPLICATION.INI

Section [bench->...]

Keyword	Value	Description
SignalAnalyzer	String	*Mandatory entry* Reference to the device entry of the signal analyzer in PHYSICAL.INI
Simulation	0 / 1	*Optional entry* Blocks simulation of all entered devices (value = 0). Enables simulation of the entered devices (value = 1). Default = 0

Keyword	Value	Description
Trace	0 / 1	*Optional entry* Blocks the tracing function (value = 0). Enables the tracing function (value = 1). Default = 0

7.1.11.4 Functions

```
Setup                                SIGANL_Setup
Library Version                      SIGANL_Lib_Version
Basic Instrument Operation
   Configure Acquisition Type        SIGANL_ConfigureAcquisitionType
   Configure Acquisition Record      SIGANL_ConfigureAcquisitionRecord
   Sample Rate                       SIGANL_SampleRate
   Actual Record Length              SIGANL_ActualRecordLength
   Configure Channel                 SIGANL_ConfigureChannel
   Configure Chan Characteristics    SIGANL_ConfigureChannelCharacteristics
   Configure Trigger                 SIGANL_ConfigureTrigger
   Configure Trigger Coupling        SIGANL_ConfigureTriggerCoupling
   Configure Edge Trigger Source     SIGANL_ConfigureEdgeTriggerSource
   Read Waveform                     SIGANL_ReadWaveform
   Initiate Acquisition              SIGANL_InitiateAcquisition
   Acquisition Status                SIGANL_AcquisitionStatus
   Abort                             SIGANL_Abort
   Fetch Waveform                    SIGANL_FetchWaveform
   TS-PAM specific functions
      Configure Coupling Relays      SIGANL_ConfigureCouplingRelays
      Configure Ground Relay         SIGANL_ConfigureGroundRelay
      Configure Trigger Output       SIGANL_ConfigureTriggerOutput
      Enable Trigger Output          SIGANL_EnableTriggerOutput
      Configure Trigger Pattern      SIGANL_ConfigureTriggerPattern
      Send Software Trigger          SIGANL_SendSoftwareTrigger
      Get Trigger Status             SIGANL_GetTriggerStatus
      Configure Scan                 SIGANL_ConfigureScan
      Actual Scan                    SIGANL_ActualScan
      Fetch Trigger                  SIGANL_FetchTrigger
Waveform Measurements
   Configure Reference Levels        SIGANL_ConfigureRefLevels
   Read Waveform Measurement         SIGANL_ReadWaveformMeasurement
   Fetch Waveform Measurement        SIGANL_FetchWaveformMeasurement
Waveform Analysis
   Find Event in Waveform            SIGANL_FindWaveformEvent
   Count Events in Waveform          SIGANL_CountWaveformEvents
   Get Waveform Value                SIGANL_GetWaveformValue
   Calculate Limit Lines             SIGANL_CalculateLimitLines
   Compare Waveform                  SIGANL_CompareWaveform
   Calculate Waveform Parameter      SIGANL_CalculateWaveformParameter
   Calculate Frequency               SIGANL_CalculateFrequency
Waveform Display
```

```
   Display Waveform in Diagram      SIGANL_ShowWaveform
   Display Waveform with Marker     SIGANL_ShowWaveformMarker
   Display Waveform with Limits     SIGANL_ShowWaveformLimits
Utility Functions
   Reset                            SIGANL_Reset
   Is Invalid Waveform Element      SIGANL_IsInvalidWfmElement
   Import Waveform Data             SIGANL_ImportWaveformData
   Export Waveform Data             SIGANL_ExportWaveformData
Cleanup                             SIGANL_Cleanup
```

7.1.12 Signal Routing Library

7.1.12.1 General

Name of the dynamic link library (DLL):	ROUTE.DLL
Name of the help file (HLP):	ROUTE.HLP
License required	R&S TS-LSRL
Supported devices:	R&S TS-PAM R&S TS-PDFT R&S TS-PFG R&S TS-PIO2 R&S TS-PMB R&S TS-PSAM R&S TS-PSM1 R&S TS-PSM2 R&S TS-PSU R&S TS-PSU12 R&S TS-PSYS1 R&S TS-PSYS2 and all other switch devices that provide an IVI-C driver of the IviSwtch class.

The Signal Routing Library makes it possible to set up complex switched connections by means of switching commands. Switched connections can be automatically routed by the analog measurement bus, i.e. the software searches for free analog measurement bus lines and automatically switches the relays in the switching path.

Extensive switched connections can also be saved under a user-specific name and then called in the test program.

Refer to chapter 8, for a detailed description of the Signal Routing Library.

NOTE:

The Signal Routing Library cannot be used together with the Switch Manager.

7.1.12.2 Entries in PHYSICAL.INI

Section [device->...]

Mandatory entry

Describes the properties of the switch modules installed in the system.

Keyword	Value	Description

Type	String	*Mandatory entry*
		for R&S TS-PAM: PAM
		for R&S TS-PDFT: PDFT
		for R&S TS-PFG: PFG
		for R&S TS-PIO2: PIO2
		for R&S TS-PMB: PMB
		for R&S TS-PSAM: PSAM
		for R&S TS-PSM1: PSM1
		for R&S TS-PSM2: PSM2
		for R&S TS-PSU: PSU
		for R&S TS-PSU12: PSU12
		for R&S TS-PSYS1: PSYS1
		for R&S TS-PSYS2: PSYS2
		for other IviSwtch compliant instruments : IVI_SWITCH

ResourceDesc	String	*Mandatory entry* VISA resource descriptor in the form: PXI[segment number]::[device number]::[function]::INSTR CAN[board]::[controller]::[frame]::[slot] GPIB[board]::[primary address]::[secondary address]

DriverPrefix	String	*Mandatory entry*
		Prefix for the IVI driver functions
		for R&S TS-PAM: rspam
		for R&S TS-PDFT: rspdft
		for R&S TS-PFG: rspfg
		for R&S TS-PIO2: rspio2
		for R&S TS-PMB: rspmb
		for R&S TS-PSAM: rspsam
		for R&S TS-PSM1: rspsm1
		for R&S TS-PSM2: rspsm2
		for R&S TS-PSU: rspsu
		for R&S TS-PSU12: rspsu
		for R&S TS-PSYS1: rspsys
		for R&S TS-PSYS2: rspsys
		for others: driver dependent

Keyword	Value	Description

DriverDLL	String	*Mandatory entry*
		File name of the driver DLL
		for R&S TS-PAM: rspam.dll
		for R&S TS-PDFT: rspdft.dll
		for R&S TS-PFG: rspfg.dll
		for R&S TS-PIO2: rspio2.dll
		for R&S TS-PMB: rspmb.dll
		for R&S TS-PSAM: rspsam.dll
		for R&S TS-PSM1: rspsm1.dll
		for R&S TS-PSM2: rspsm2.dll
		for R&S TS-PSU: rspsu.dll
		for R&S TS-PSU12: rspsu.dll
		for R&S TS-PSYS1: rspsys.dll
		for R&S TS-PSYS2: rspsys.dll
		for others: driver dependent

DriverOption	String	*Optional entry* Option string being passed to the device driver during the driver's InitWithOptions function. See the online help file for the appropriate device driver. **NOTE:** **For R&S TS-PMB modules, the option "DriverSet-up=CRAuto:1" must not be used.**

Section [device->ABUS]

Mandatory entry

Device section for the analog bus.

Keyword	Value	Description
Type	String	*Mandatory entry* AB

Section [io_channel->system]

Optional entry

The system channel table contains a list of user-specific channel names which are assigned to the physical device names and to the physical device channel names. The defined names apply to all test applications running on the system.

Keyword	Value	Description
<logical channel name>	String	Physical channel description in the combination <device name>!<physical device channel name>

7.1.12.3 Entries in APPLICATION.INI

Section [bench->...]

Mandatory entry

Contains a list of switch devices, options and links to the channel table and switch settings.

Keyword	Value	Description
SwitchDevice<i>	String	*Mandatory entry* Refers to a device entry section of a switch device in PHYSICAL.INI. <i> stands for a number from 1,2,3,…,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in case it is 1.
AnalogBus	String	*Mandatory entry* Refers to the device section of the analog bus in PHYSICAL.INI.
AppChannelTable	String	*Mandatory entry* Refers to a section [io_channel->…] with defined channel names in APPLICATION.INI.
SwitchSettings	String	*Optional entry* Refers to a section [switch->…] with defined switch settings in APPLICATION.INI

Keyword	Value	Description
Simulation	0 / 1	*Optional entry* Blocks the simulation of the entered devices (value = 0). Enables simulation of the entered devices (value = 1). Default = 0
Trace	0 / 1	*Optional entry* Blocks the tracing function of the library (value = 0). Enables the tracing function of the library (value = 1). Default = 0
ChannelTableCaseSensitive	0 / 1	*Optional entry* The channel names in the channel table are treated case-sensitive (value = 1) or case-insensitive (value = 0). Default = 0
SignalRoutingDisplay	0 / 1	*Optional entry* Displays a window with actual signal routing information (value=1). Default = 0

Section [io_channel->...]

Mandatory entry

Contains a list of user-specific channel names which are assigned to the physical device names and to the physical device channel names. The defined names apply only to the relevant application. For details about channel name syntax see chapter 8.3.4.

Keyword	Value	Description
<logical channel name>	String	Physical channel description in the combination <device name>!<physical device channel name>
<logical channel name>	String	Logical channel name from the section [io_channel->system] from PHYSICAL.INI.

Section [switch->…]

Optional entry

Contains a list of user-specific switch setting names which are assigned to signal routing command strings. Refer to chapter 8.4.3 for details.

Keyword	Value	Description
#<switch setting name>	String	Signal routing command

7.1.12.4 Functions

```
Setup                                ROUTE_Setup
Library Version                      ROUTE_Lib_Version
Signal Routing
   Execute                           ROUTE_Execute
Cleanup                              ROUTE_Cleanup
```

7.1.13 Switch Manager Library

7.1.13.1 General

Name of the dynamic link library (DLL):	SWMGR.DLL
Name of the help file (HLP):	SWMGR.HLP
License required	R&S TS-LBAS
Supported devices:	R&S TS-PAM R&S TS-PDFT R&S TS-PFG R&S TS-PIO2 R&S TS-PMA R&S TS-PMB R&S TS-PRL0 R&S TS-PRL1 R&S TS-PSAM R&S TS-PSM1 R&S TS-PSM2 R&S TS-PSU R&S TS-PSU12 and all other switch devices that provide an IVI-C driver of the IviSwtch class.

The Switch Manager Library provides functions for the switching of signals. It controls the device drivers of the relevant switching modules (e.g. R&S TS-PRL1, R&S TS-PMA). The Test Library changes switch requests with channel names (standard I/O channels or system/application specific channels) into calls to the device drivers of the existing switch modules.

7.1.13.2 Entries in PHYSICAL.INI

Section [device->...]

Mandatory entry

Describes the properties of the switch cards installed in the system.

Keyword	Value	Description

Type	String	*Mandatory entry*
		pam = R&S TS-PAM Analyzer Module
		pdft = R&S TS-PDFT Digital Functional Test Module
		pfg = R&S TS-PFG Function Generator Module
		pio2 = R&S TS-PIO2 Analog/Digital IO Module 2
		pma1 = R&S TS-PMA Matrix Module with R&S TS-PMA1 Relay Modules
		pma2 = R&S TS-PMA Matrix Module with R&S TS-PMA2 Relay Modules
		pmb = R&S TS-PMB Matrix Module
		prl0 = R&S TS-PRL0 Universal Relay Module
		prl1 = R&S TS-PRL1 Universal Relay Module
		psam = R&S TS-PSAM Analog Source and Measurement Module
		psm1 = R&S TS-PSM1 Power Switch Module
		psm2 = R&S TS-PSM2 Multiplex/Switch Module 2
		psu = R&S TS-PSU Power Supply/Load Module
		psu12 = R&S TS-PSU12 Power Supply/Load Module 12V
		ivi_switch = any other switching module or any other switchpanel card with an IVI device driver.

ResourceDesc	String	*Mandatory entry* VISA device properties and device description in the form: PXI[segment number]::[device number]::[function]::INSTR CAN[board]::[controller]::[frame]::[slot]

Keyword	Value	Description
DriverPrefix	String	*Mandatory entry* Prefix for the IVI driver functions R&S TS-PAM: rspam R&S TS-PDFT: rspdft R&S TS-PFG: rspfg R&S TS-PIO2: rspio2 R&S TS-PMA: rspma R&S TS-PMB: rspmb R&S TS-PRL0: rsprl0 R&S TS-PRL1: rsprl1 R&S TS-PSAM: rspsam R&S TS-PSM1: rspsm1 R&S TS-PSM2: rspsm2 R&S TS-PSU: rspsu R&S TS-PSU12: rspsu Other designations: Dependent on the drivers used
Driver DLL	String	*Mandatory entry* File name of the driver DLL R&S TS-PAM: rspam.dll R&S TS-PDFT: rspdft.dll R&S TS-PFG: rspfg.dll R&S TS-PIO2: rspio2.dll R&S TS-PMA: rspma.dll R&S TS-PMB: rspmb.dll R&S TS-PRL0: rsprl0.dll R&S TS-PRL1: rsprl1.dll R&S TS-PSAM: rspsam.dll R&S TS-PSM1: rspsm1.dll R&S TS-PSM2: rspsm2.dll R&S TS-PSU: rspsu.dll R&S TS-PSU12: rspsu.dll Other designations: Dependent on the drivers used
DriverOption	String	*Optional entry* Optional indications which are passed to the device driver during the Driver_Init function. See the online help of the relevant switch device drivers.

Section [device->ABUS]

Mandatory entry

Device section for the analog bus.

Keyword	Value	Description
Type	String	ab = Analog Bus

Section [io_channel->system]

Optional entry

Contains a list of user-specific channel names which are assigned to the physical device names and to the physical device channel names. The defined names apply to all test applications running on the system.

Keyword	Value	Description
<user-defined name>	String	Physical channel description in the combination <device name>!<device channel name>

7.1.13.3 Entries in APPLICATION.INI

Section [bench->...]

Mandatory entry

Contains a list of switch devices

Keyword	Value	Description
SwitchDevice<i>	String	*Mandatory entry* Refers to a section with switch devices in PHYSICAL.INI <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.
AnalogBus	String	*Optional entry* Refers to the device section of the analog bus in PHYS-ICAL.INI.
AppChannelTable	String	*Optional entry* Refers to a section with defined channel names in AP-PLICATION.INI.

Keyword	Value	Description
Simulation	0 / 1	*Optional entry* Blocks simulation of all entered devices (value = 0). Enables simulation of the entered devices (value = 1).
Trace	0 / 1	*Optional entry* Blocks the tracing function (value = 0), enables the tracing function (value = 1). Default = 0
ChannelTableCase Sensitive	0 / 1	*Optional entry* The channel names in the channel table are treated case-sensitive (value = 1) or case-insensitive (value = 0).

Section [io_channel->...]

Optional entry

Contains a list of user-specific channel names which are assigned to the physical device names and to the physical device channel names. The defined names apply only to the relevant application. For details about channel name syntax see chapter 8.3.4.

Keyword	Value	Description
<user-defind name>	String	Physical channel description in the combination <device name>!<device channel name>
<user-defind name>	String	Channel names from the list [io_channel->system] from the PHYSICAL.INI.

7.1.13.4 Functions

```
Management
         Setup                        SWMGR_Setup
         Library Version              SWMGR_Lib_Version
         Cleanup                      SWMGR_Cleanup
Configuration Functions
   Configure Coupling Mode            SWMGR_ConfigureCouplingMode
   Configure Coupling Relays          SWMGR_ConfigureCouplingRelays
Route Functions
         Connect Channels             SWMGR_Connect
         Disconnect Channels          SWMGR_Disconnect
         Disconnect All Channels      SWMGR_DisconnectAll
         Switch Is Debounced?         SWMGR_IsDebounced
         Wait For Debounce            SWMGR_WaitForDebounce
```

7.1.14 Toolkit TestStand Library

7.1.14.1 General

Name of the dynamic link library (DLL):	TOOLKITTESTSTAND.DLL
Name of the help file (HLP):	TOOLKITTESTSTAND.HLP
License required	R&S TS-LBAS
Supported devices:	not usable

ToolkitTestStand is a debugging tool which allows to access the run-time state information of the radio communication tester in TestStand. The necessary TestStand data types are defined in the demo sequence „ToolkitTestStand_Demo.seq“.

For each memory block, there are three functions to:

- Read (C structure --> TestStand data structure)
- Compare (Trace the differences between the C structure & TestStand data structure)
- Write (TestStand data structure --> C structure)

7.1.14.2 Entries in PHYSICAL.INI

No entries

7.1.14.3 Entries in APPLICATION.INI

No entries

7.1.14.4 Functions

```
Library Version                  TKTS_Lib_Version
Memory Block Handoff
   ReadSharedMemoryBlockHandoff   TKTS_ReadSharedMemoryBlockHandoff
   CompareSharedMemoryBlockHandoffTKTS_CompareSharedMemoryBlockHandoff
   WriteSharedMemoryBlockHandoff  TKTS_WriteSharedMemoryBlockHandoff
Memory Block DNA
   ReadSharedMemoryBlockDNA       TKTS_ReadSharedMemoryBlockDNA
   CompareSharedMemoryBlockDNA    TKTS_CompareSharedMemoryBlockDNA
   WriteSharedMemoryBlockDNA      TKTS_WriteSharedMemoryBlockDNA
Memory Block AUDIO
   ReadSharedMemoryBlockAUDIO     TKTS_ReadSharedMemoryBlockAUDIO
   CompareSharedMemoryBlockAUDIO  TKTS_CompareSharedMemoryBlockAUDIO
   WriteSharedMemoryBlockAUDIO    TKTS_WriteSharedMemoryBlockAUDIO
```

```
Memory Block BLUETOOTH
   ReadSharedMemoryBlockBT        TKTS_ReadSharedMemoryBlockBT
   CompareSharedMemoryBlockBT     TKTS_CompareSharedMemoryBlockBT
   WriteSharedMemoryBlockBT       TKTS_WriteSharedMemoryBlockBT
Analyse log file
   Scan LogFile for a variable    TKTS_ScanLogFile
   Conf Param, give SubSeqNumber  TKTS_TestNumConfParamLogFile
   Count the number of lines      TKTS_NumLineLogFile
   Extract data from a logFile    TKTS_FilterLogFile
```

7.1.15 Utility Library

7.1.15.1 General

Name of the dynamic link library (DLL):	UTIL.DLL
Name of the help file (HLP):	UTIL.HLP
License required	R&S TS-LBAS
Supported devices:	not usable

Miscellaneous utility functions.

7.1.15.2 Entries in PHYSICAL.INI

No entries

7.1.15.3 Entries in APPLICATION.INI

No entries

7.1.15.4 Functions

`Library Version`	`UTIL_Lib_Version`
`Information`	
`GTSL Version`	`UTIL_GTSL_Version`
`GTSL Registry Value`	`UTIL_GTSL_Registry_Value`
`Time Functions`	
`Delay (obsolete)`	`UTIL_Delay`
`High-resolution Timer`	
`Sleep`	`UTIL_Hrestim_Sleep`
`Sleep Until`	`UTIL_Hrestim_Sleep_Until`
`Get Time Stamp`	`UTIL_Hrestim_Time_Stamp`
`Get Timer Resolution`	`UTIL_Hrestim_Resolution`
`TSVP Module Information`	
`Module Search`	`UTIL_Module_Search`
`Sort Module List`	`UTIL_Module_Sort_List`
`Get Attribute (Integer)`	`UTIL_Module_Get_Attribute_Int`
`Get Attribute (String)`	`UTIL_Module_Get_Attribute_String`
`Free Module List`	`UTIL_Module_Free_List`
`HTML Help`	
`Show HTML Help`	`UTIL_Show_Html_Help`
`Close HTML Help`	`UTIL_Close_Html_Help`

7.2 In-Circuit Test Libraries

☞

NOTE:

For further information on the In-Circuit-Test, R&S EGTSL and R&S IC-Check see “Software Description Enhanced Generic Test Software Library R&S EGTSL” and “Software Description Generic Test Software Library R&S IC-Check”.

7.2.1 IC-Check Library

7.2.1.1 General

Name of the dynamic link library (DLL):	ICCHECK.DLL
Name of the help file (HLP):	ICCHECK.HLP
License required	R&S TS-LBAS and R&S TS-LICC
Supported devices:	R&S TS-PMB Matrix Module R&S TS-PSAM Source and Measurement Module

The IC-Check Test Library offers functions for the IC check using the R&S GTSL software and the R&S TS-PSAM and R&S TS-PMB modules. The functions allow to

- load, run and debug ICC programs
- generate a report

7.2.1.2 Entries in PHYSICAL.INI

Section [device->…]

Keyword	Value	Description

Type	String	*Mandatory entry*
		pmb = R&S TS-PMB Matrix Module
		psam = R&S TS-PSAM Source and Measurement Module

ResourceDesc	String	*Mandatory entry* VISA resource descriptor in the form PXI[segment number]::[device number]::[function]::INSTR CAN[board]::[controller]::[frame]::[slot]

Keyword	Value	Description
DriverPrefix	String	*Mandatory entry* prefix for the IVI driver functions, without underscore R&S TS-PMB: rspmb R&S TS-PSAM: rspsam
DriverDll	String	*Mandatory entry* File name of the driver DLL R&S TS-PMB: rspmb.dll R&S TS-PSAM: rspsam.dll
DriverOption	String	*Optional entry* Option string being passed to the device driver during the driver's InitWithOptions function. See the online help file for the appropriate device driver.

7.2.1.3 Entries in APPLICATION.INI

Section [bench->…]

Keyword	Value	Description
ICCDevice	String	*Mandatory entry* Refers to the device section of the R&S TS-PSAM
SwitchDevice<i>	String	*Mandatory entry* Refers to a device sections of a switch device R&S TS-PMB in PHYSICAL.INI. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.
AppChannelTable	String	*Mandatory entry* Refers to a section with defined channel names in APPLICATION.INI.
Simulation	0 / 1	*Optional entry* Blocks the simulation of the entered devices (value = 0). Enables simulation of the entered devices (value = 1). Default = 0
Trace	0 / 1	*Optional entry* Blocks the tracing function of the library (value = 0). Enables the tracing function of the library (value = 1). Default = 0

Keyword	Value	Description
ChannelTableCase-Sensitive	0 / 1	Optional entry The channel names in the channel table are treated case-sensitive (value = 1) or case-insensitive (value = 0).

Section [io_channel->...]

Contains a list of user-specific channel names (or ATG-defined channel names) which are assigned to the physical device names and to the physical device channel names. The defined names apply only to the relevant application. For details about channel name syntax see chapter 8.3.4.

Keyword	Value	Description
<user-defined name>	String	Physical channel description in the form: <device name>!<device channel name>.

7.2.1.4 Functions

Setup	ICCHECK_Setup
Cleanup	ICCHECK_Cleanup
Library Version	ICCHECK_Lib_Version
Program Control	
Load Program	ICCHECK_Load_Program
Run Program	ICCHECK_Run_Program
Debug Program	ICCHECK_Debug_Program
Report Generation	
Write Report to File	ICCHECK_Write_Report
Load Detailed Report	ICCHECK_Load_Detailed_Report
Get Detailed Report Entry	ICCHECK_Get_Detailed_Report_Entry
Attribute Information	
Get Attribute Int	ICCHECK_Get_Attribute_Int
Get Attribute Real	ICCHECK_Get_Attribute_Real
Get Attribute String	ICCHECK_Get_Attribute_String

7.2.2 In-Circuit-Test Library

7.2.2.1 General

Name of the dynamic link library (DLL):	ICT.DLL
Name of the help file (HLP):	ICT.HLP
License required	R&S TS-LBAS and R&S TS-LEGT or R&S TS-LEG2
Supported devices:	R&S TS-PICT ICT Extension Module R&S TS-PMB Matrix Module R&S TS-PSAM Source and Measurement Module R&S TS-PSU Power Supply / Load Module R&S TS-PSU12 Power Supply / Load Module 12V

The in-circuit test library offers functions for the in-circuit test using the R&S GTSL software and the R&S TS-PSAM, R&S TS-PICT, R&S TS-PSU, R&S TS-PSU12 and R&S TS-PMB modules.

The functions allow to

- load, run and debug ICT programs
- load limit files
- generate a report

7.2.2.2 Entries in PHYSICAL.INI

Section [device->...]

Keyword	Value	Description

Type	String	*Mandatory entry*
		pict = R&S TS-PICT ICT Extension Module
		pmb = R&S TS-PMB Matrix Module
		psam = R&S TS-PSAM Source and Measurement Module
		psu = R&S TS-PSU Power Supply/Load Module
		psu12 = R&S TS-PSU12 Power Supply/Load Module 12V

ResourceDesc	String	*Mandatory entry* VISA resource descriptor in the form PXI[segment number]::[device number]::[function]::IN-STR CAN[board]::[controller]::[frame]::[slot]
DriverPrefix	String	*Mandatory entry* Prefix for the IVI driver functions, without underscore: R&S TS-PICT : rspict R&S TS-PMB : rspmb R&S TS-PSAM : rspsam R&S TS-PSU: rspsu R&S TS-PSU12: rspsu
DriverDLL	String	*Mandatory entry* File name of the driver DLL R&S TS-PICT : rspict.dll R&S TS-PMB : rspmb.dll R&S TS-PSAM : rspsam.dll R&S TS-PSU : rspsu.dll R&S TS-PSU12 : rspsu.dll
DriverOption	String	*Optional entry* Option string being passed to the device driver during the Driver_Init function. See the online help file for the appropriate device driver.

7.2.2.3 Entries in APPLICATION.INI

Section [bench->...]

Keyword	Value	Description
ICTDevice1	String	*Mandatory entry* Refers to the device section of the R&S TS-PSAM
ICTDevice2	String	*Optional entry* Refers to the device section of the R&S TS-PICT or R&S TS-PSU / R&S TS-PSU12
ICTDevice3	String	*Optional entry* Refers to the device section of the R&S TS-PICT or R&S TS-PSU / R&S TS-PSU12
SwitchDevice<i>	String	*Mandatory entry* Refers to a section with switch devices in PHYSICAL.INI. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.
AppChannelTable	String	*Mandatory entry* Refers to a section with defined channel names in APPLICATION.INI.
Simulation	0 / 1	*Optional entry* Blocks the simulation of the entered devices (value = 0). Enables simulation of the entered devices (value = 1). Default = 0
Trace	0 / 1	*Optional entry* Blocks the tracing function of the library (value = 0). Enables the tracing function of the library (value = 1). Default = 0
ChannelTableCase Sensitive	0 / 1	*Optional entry* The channel names in the channel table are treated case-sensitive (value = 1) or case-insensitive (value = 0).

Section [io_channel->...]

Contains a list of user-specific channel names (or ATG-defined channel names) which are assigned to the physical device names and to the physical device channel names. The defined names apply only to the relevant application. For details about channel name syntax see chapter 8.3.4.

Keyword	Value	Description
<user-defined name>	String	Physical channel description in the combination <device name>!<device channel name>

7.2.2.4 Functions

```
Setup                              ICT_Setup
Library Version                    ICT_Lib_Version
EGTSL Runtime Version              ICT_Runtime_Version
Program Control
   Load Program                    ICT_Load_Program
   Run Program                     ICT_Run_Program
   Debug Program                   ICT_Debug_Program
   Unload Program                  ICT_Unload_Program
Report Generation
   Write Report to File            ICT_Write_Report
   Load Detailed Report            ICT_Load_Detailed_Report
   Get Detailed Report Entry       ICT_Get_Detailed_Report_Entry
   Get Detailed Report Entry (Extended)
                                   ICT_Get_Detailed_Report_Entry_Ex
   Get TestStand Report Entry      ICT_Get_TestStand_Report_Entry
   Transfer Report to QUOTIS       ICT_Transfer_Quotis_Report
Limit Loader
   Load Limits                     ICT_Load_Limits
Error Handling
   Get Error Log                   ICT_Get_Error_Log
Cleanup                            ICT_Cleanup
```

7.2.3 Vacuum Control Library

7.2.3.1 General

Name of the dynamic link library (DLL):	VACUUM.DLL
Name of the help file (HLP):	VACUUM.HLP
License required:	R&S TS-LBAS
Supported devices:	R&S TS-PSYS1, System Module R&S TS-PSYS2, System Module

The Vacuum Library offers functions for one or more vacuum control units R&S TS-PVAC.

7.2.3.2 Entries in PHYSICAL.INI

Section [device->...]

Keyword	Value	Description

Type	String	*Mandatory entry*
		psys1 = R&S TS-PSYS1, System Module
		psys2 = R&S TS-PSYS2, System Module

ResourceDesc	String	*Mandatory entry* resource descriptor in the form CAN[board]::[controller]::[frame]::[slot]
DriverPrefix	String	*Mandatory entry* Prefix for the IVI driver functions, without underscore: R&S TS-PSYS1, R&S TS-PSYS2 : rspsys
DriverDLL	String	*Mandatory entry* File name of the driver DLL R&S TS-PSYS1, R&S TS-PSYS2: rspsys.dll
DriverOption	String	*Optional entry* Option string being passed to the device driver during the Driver_Init function. See the online help file for the appropriate device driver.

7.2.3.3 Entries in APPLICATION.INI

Section [bench->...]

Keyword	Value	Description
VacuumControl<i>	String	*Mandatory entry* Refers to a section with devices in PHYSICAL.INI <i> stands for a number from 1,2,3,...,40. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.
Simulation	0 / 1	*Optional entry* Blocks the simulation of the entered devices (value = 0). Enables simulation of the entered devices (value = 1). Default = 0
Trace	0 / 1	*Optional entry* Blocks the tracing function of the library (value = 0), enables the tracing function of the library (value = 1). Default = 0

7.2.3.4 Functions

```
Management
   Setup                          VACUUM_Setup
   Library Version                VACUUM_Lib_Version
   Cleanup                        VACUUM_Cleanup

Control
   Control                        VACUUM_Control
Status
   Status                         VACUUM_Status
```

7.3 Radio Communication Libraries

7.3.1 Analog Test Library

7.3.1.1 General

Name of the dynamic link library (DLL):	ANALOG.DLL
Name of the help file (HLP):	ANALOG.HLP
License required	R&S TS-LANL
Supported devices:	Radio Communication Tester CMD80, CMU200

The Analog Test Library provides test functions for the function test and final test of mobile telephones. The Test Library contains functions for

- opening, configuring and closing the Radio Communication Tester present in the test system.
- carrying out measurements with the Radio Communication Tester present in the test system.

7.3.1.2 Entries in PHYSICAL.INI

Section [device->...]

Keyword	Value	Description
Type	String	*Mandatory entry* CMD80 or CMU200
ResourceDesc	String	*Mandatory entry* VISA resource descriptor in the form GPIB[board]::primaryAddress[::secondaryAddress]
ResourceDesc_ RF_NSig	String	(CMU only) VISA resource descriptor for the RF non-signaling part
ResourceDesc_ AMPS_Nsig	String	(CMU only) VISA resource descriptor for the AMPS non-signaling part
ResourceDesc_ AMPS_Sig	String	(CMU only) VISA resource descriptor for the AMPS signaling part
ResourceDesc_Audi o	String	(CMU only) VISA resource descriptor for the audio part

Keyword	Value	Description
Trace	0 / 1	*Optional entry* Enables/disables CMU driver-level tracing
TraceFile	String	*Optional entry* Defines the path and the name of the trace file. Default = „“

7.3.1.3 Entries in APPLICATION.INI

Section [bench->...]

Keyword	Value	Description
RadioComTester	String	*Mandatory entry* Reference to the device entry of the Radio Communication Tester in PHYSICAL.INI.
Simulation	0 / 1	*Optional entry* Blocks the simulation of the entered devices (value = 0). Enables simulation of the entered devices (value = 1). Default = 0
Trace	0 / 1	*Optional entry* Blocks the tracing function (value = 0), enables the tracing function (value = 1). Default = 0
ViewSharedMem-Blocks	0 / 1	*Optional entry* Enables/disables extended tracing. Default = 0

Optional entries

Entries for calibration.

Keyword	Value	Description
Calibration	0 / 1	Blocks the calibration function (value = 0), enables the calibration function (value = 1).
CalibrationFile<i>	String	Path and file name of the calibration files. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.

Keyword	Value	Description
CalibrationPath_ ANAG_Tx<i>	String	List of calibration tables for the ANAG-Tx path. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.
CalibrationPath_ ANAG_Rx<i>	String	List of calibration tables for the ANAG-Rx path. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.
CalibrationPath_ AUDIO_Ear<i>	String	List of calibration tables for the audio ear path. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.
CalibrationPath_ AUDIO_Mouth<i>	String	List of calibration tables for the audio mouth path. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.

7.3.1.4 Functions

```
Setup                                  ANAG_Setup
Library Version                        ANAG_Lib_Version
Non Signalling
   Configuration
      RF Analyser Settings             ANAG_NSig_Conf_RF_Analyser
      RF Generator Settings            ANAG_NSig_Conf_RF_Generator
      RF Generator On/Off              ANAG_NSig_Conf_RF_Gen_OnOff
   SAT Generator
      Settings                         ANAG_NSig_Conf_SATG_Settings
      SAT Generator On/Off             ANAG_NSig_Conf_SATG_OnOff
   ST Generator
      Settings                         ANAG_NSig_Conf_STG_Settings
      ST Generator On/Off              ANAG_NSig_Conf_STG_OnOff
Signalling
   Configuration
      BTS Parameters
      Global Settings                  ANAG_Sig_Conf_BS_RF_Settings
      Voice Channel MAC                ANAG_Sig_Conf_BS_VMAC
      Voice Channel                    ANAG_Sig_Conf_BS_VCH
      Voice Channel Level              ANAG_Sig_Conf_BS_VCH_Level
      SAT Color Code                   ANAG_Sig_Conf_BS_SCC
      Network ID
         Network Parameters            ANAG_Sig_Conf_Netw_Parm
      Handoff Transition
         Handoff from Analog           ANAG_Sig_Conf_Handoff_Analog
         Handoff BTS Parameter          ANAG_Sig_Conf_Handoff_BS_Param
      Call Established
         Voice Channel MAC              ANAG_Sig_Conf_CEST_VMAC
         Voice Channel                  ANAG_Sig_Conf_CEST_VCH
         Voice Channel Level            ANAG_Sig_Conf_CEST_VCH_Level
         SAT Color Code                 ANAG_Sig_Conf_CEST_SCC
   Call Management
      Call Establishment                ANAG_Sig_Call
      Release Call                      ANAG_Sig_Call_Release
      Registration                      ANAG_Sig_Call_Registration
      MS Identification                 ANAG_Sig_Call_MS_ID
      Call Status                       ANAG_Sig_Call_Status
      Execute Handoff                   ANAG_Sig_HandOff
Common Generators
   Modulation Generator
      Settings                          ANAG_Conf_ModG
      Generator On/Off                  ANAG_Conf_ModG_OnOff
   AF Generator
      Settings                          ANAG_Conf_AFG
      Generator On/Off                  ANAG_Conf_AFG_OnOff
Measurement Objects
   Custom measurements                  ANAG_Custom_Measurements
   Rx Receiver Tests
      AF Analyser
         Configurations
            Control / Limits            ANAG_Conf_Rx_AFA
            Filter                      ANAG_Conf_Rx_AFA_Filter
            Read                        ANAG_Read_Rx_AFA
            Fetch                       ANAG_Fetch_Rx_AFA
      Hum and Noise
         Configurations
            Control / Limits            ANAG_Conf_Rx_HumNoise
            Filter                      ANAG_Conf_Rx_HumNoise_Filter
         Read                           ANAG_Read_Rx_HumNoise
         Fetch                          ANAG_Fetch_Rx_HumNoise
      Sensitivity
         Configurations
            Control / Limits            ANAG_Conf_Rx_Sens
            Filter                      ANAG_Conf_Rx_Sens_Filter
         Read                           ANAG_Read_Rx_Sens
         Fetch                          ANAG_Fetch_Rx_Sens
      Audio Freq Response
         <Create Class or Function Panel Window>
   Tx Transmitter Tests
      Modulation
         Configurations
            Control / Limits            ANAG_Conf_Tx_Mod
         Read
            Read All Deviations
               AMPS/TACS/ETACS/JTACS
                  Read All Deviations  ANAG_Read_Tx_Mod_AMPS
```

NAMPS/NTACS	
Read All Deviations	ANAG_Read_Tx_Mod_NAMPS
Read ST Deviation	ANAG_Read_Tx_Mod_ST
Fetch	
Fetch All Deviations	ANAG_Fetch_Tx_Mod_AMPS
Fetch ST Deviation	ANAG_Fetch_Tx_Mod_ST
Peak Power	
Read	ANAG_Read_Tx_Peak_Power
Hum and Noise	
Configurations	
Control / Limits	ANAG_Conf_Tx_HumNoise
Read	ANAG_Read_Tx_HumNoise
Fetch	ANAG_Fetch_Tx_HumNoise
AF Level Search	
Configurations	
Control / Limits	ANAG_Conf_Tx_AFLevSearch
Filter	ANAG_Conf_Tx_AFLevSearch_Filter
Read	ANAG_Read_Tx_AFLevSearch
Fetch	ANAG_Fetch_Tx_AFLevSearch
Audio Freq Response	
<Create Class or Function Panel Window>	
RCT General Configurations	
Network Selection	ANAG_Conf_Network
Test Mode	ANAG_Conf_Test_Mode
Input Output	ANAG_Conf_IO
Display On/Off	ANAG_Conf_Display
Utility	
RCT_Booked	ANAG_RCT_Booked
RCT_Vacated	ANAG_RCT_Vacated
RCT_Get_Sharing_State	ANAG_RCT_Get_Sharing_State
Change Calibration Paths	ANAG_Util_Conf_Calibration
Get Calibration Offsets	ANAG_Util_GetCalibrationOffsets
Clean Error Queue	ANAG_Util_CleanErrorQueue
Cleanup	ANAG_Cleanup

7.3.2 Audio Test Library

7.3.2.1 General

Name of the dynamic link library (DLL):	AUDIO.DLL
Name of the help file (HLP):	AUDIO.HLP
License required	R&S TS-LBAS
Supported devices:	Radio Communication Tester CMU200, Audio Analyzer UPL

The Audio Test Library provides audio test functions for the function test and final test of mobile telephones.

The test library contains functions for

- Single Tone and Multitone audio measurements

7.3.2.2 Entries in PHYSICAL.INI

Section [device->...]

Keyword	Value	Description

Type	String	*Mandatory entry*
		CMU = Radio Communication Tester CMU200
		UPL = Audio analyzer UPL

ResourceDesc	String	*Mandatory entry* VISA resource descriptor in the form GPIB[board]::primaryAddress[::secondaryAddress]
ResourceDesc_Audio	String	*Mandatory entry, CMU only* VISA resource descriptor for the audio component
Trace	0 / 1	*Optional entry, CMU only* Blocks the tracing function of the driver (value = 0), enables the tracing function of the driver (value = 1). Default = 0
TraceFile	String	*Optional entry, CMU only* Defines the path and the name of the trace file. Default = “”

7.3.2.3 Entries in APPLICATION.INI

Section [bench->...]

Keyword	Value	Description
AudioTester	String	*Mandatory entry* Reference to the device entry of the AudioTester in PHYSICAL.INI.
Simulation	0 / 1	*Optional entry* Blocks the simulation of the entered devices (value = 0). Enables simulation of the entered devices (value = 1). Default = 0
Trace	0 / 1	*Optional entry* Blocks the tracing function of the library (value = 0), enables the tracing function of the library (value = 1). Default = 0
ViewSharedMem-Blocks	0 / 1	*Optional entry* Enables/disables extended tracing. Default = 0

Optional entries

Entries for calibration.

Keyword	Value	Description
Calibration	0 / 1	Blocks the calibration function (value = 0), enables the calibration function (value = 1).
CalibrationFile<i>	String	Path and file name of the calibration files. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.
CalibrationPath_ AUDIO_Ear<i>	String	List of calibration tables for the audio ear path. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.
CalibrationPath_ AUDIO_Mouth<i>	String	List of calibration tables for the audio mouth path. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.

7.3.2.4 Functions

Setup	AUDIO_Setup
Library Version	AUDIO_Lib_Version
Configuration	
Generator	AUDIO_AfConfGeneratorSettings
Analyzer	AUDIO_AfConfAnalyzerSettings
Switch AF Generator ON OFF	AUDIO_AfConfGeneratorOnOff
Speech Coder Options	AUDIO_AfConfSpeechCoder
Multitone	
Set Tone Definition	AUDIO_AfConfMultitoneToneDef
Set Analyzer Levels	AUDIO_AfConfMultitoneAnaLvl
Set Limits	AUDIO_AfConfMultitoneLimits
Measurements	
SingleTone	AUDIO_AfMeasVoltage
Multitone	
Level Array	AUDIO_AfReadMultitoneLvlArray
Matching Array	AUDIO_AfReadMultitoneMatchArray
Utility	
Calibration	
Change path	AUDIO_UtilConfCalibration
Get Calibration	AUDIO_UtilGetCalibrationOffset
Cleanup	AUDIO_Cleanup

7.3.3 Bluetooth Test Library

7.3.3.1 General

Name of the dynamic link library (DLL):	BLUETOOTH.DLL
Name of the help file (HLP):	BLUETOOTH.HLP
License required	R&S TS-LBT
Supported devices:	CMU200

The Bluetooth Test Library provides convenient access to all test scenarios needed to perform the RX/TX measurements according to Core Spec. Part I:1.

The Test Library contains functions for

- opening, configuring and closing the Radio Communication Tester present in the test system.
- carrying out measurements with the Radio Communication Tester present in the test system.

7.3.3.2 Entries in PHYSICAL.INI

Section [device->...]

Keyword	Value	Description
Type	String	*Mandatory entry* CMU
ResourceDesc	String	*Mandatory entry* VISA resource descriptor in the form GPIB[board]::primaryAddress[::secondaryAddress]
ResourceDesc_Bluetooth_Nsig	String	VISA resource descriptor for the Bluetooth non-signaling part
ResourceDesc_Bluetooth_Sig	String	VISA resource descriptor for the Bluetooth signaling part
Trace	0 / 1	*Optional entry* Enables/disables CMU driver-level tracing
TraceFile	String	*Optional entry* Filename for CMU driver trace

7.3.3.3 Entries in APPLICATION.INI

Section [bench->...]

Keyword	Value	Description
RadioComTester	String	*Mandatory entry* Reference to radio communication device entry
Simulation	0 / 1	*Optional entry* Enables/disables library-level simulation, default = 0
Trace	0 / 1	*Optional entry* Enables/disables tracing, default = 0
ViewSharedMem-Blocks	0 / 1	*Optional entry* Enables/disables extended tracing, default = 0

Calibration entries (Optional entries)

Keyword	Value	Description
Calibration	0 / 1	Enables/disables calibration
CalibrationFile<i>	String	Path and file name of the calibration files. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.
CalibrationPath_BT_Tx<i>	String	List of calibration tables for the Bluetooth Tx path. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.
CalibrationPath_BT_Rx<i>	String	List of calibration tables for the. Bluetooth Rx path. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.

7.3.3.4 Functions

Management	
Setup	BT_Setup
Library Version	BT_Lib_Version
Cleanup	BT_Cleanup
Non Signalling Mode	
RF Generator Settings	BT_NSig_Conf_RF_Generator_Settings
RF Generator On/Off	BT_NSig_Conf_RF_Gen_OnOff
Signalling Mode	
Configuration	

Network Parameters	
Master Signalling	
MS global Settings	BT_Sig_Conf_MS_Settings
MS BD_Address	BT_Sig_Conf_MS_BDAddress
TX Level	BT_Sig_Conf_MS_TX_Level
Paging	
Paging global Settings	BT_Sig_Conf_Netw_Pag_Settings
DUT BD_Address	BT_Sig_Conf_Netw_Pag_BDAddress
Test Mode	
Test Mode Settings	BT_Sig_Conf_Netw_Test_Settings
Transmitter Test	BT_Sig_Conf_TM_TX_Test
Loopback Test	BT_Sig_Conf_TM_RX_Test
Call Established	
Test Mode Settings	BT_Sig_Conf_Test_Settings
Call Management	
Call Inquiry	BT_Sig_Call_Inquiry
Call Connection	BT_Sig_Call_Connection
Call Status	BT_Sig_Call_Status
Call Detach	BT_Sig_Call_Detach
Custom measurements	BT_Sig_Custom_Meas
Transmitter Tests	
Power vs Time	
Configuration	
Measurement Control	BT_Sig_Conf_PVT
Results	
Fetch	BT_Sig_Fetch_PVT
Read	BT_Sig_Read_PVT
Read Multichannel	BT_Sig_Read_PVT_Multichan
Fetch Multichannel	BT_Sig_Fetch_PVT_Multichan
Modulation	
Configuration	
Measurement Control	BT_Sig_Conf_Mod
Results	
Fetch	BT_Sig_Fetch_Mod
Read	BT_Sig_Read_Mod
Read Multichannel	BT_Sig_Read_Mod_Multichan
Fetch Multichannel	BT_Sig_Fetch_Mod_Multichan
Receiver Tests	
BER	
Configuration	
Measurement Control	BT_Sig_Conf_BER_Control
Loopback Parameters	BT_Sig_Conf_BER_Settings
Fetch	BT_Sig_Fetch_BER
Read	BT_Sig_Read_BER
BER Search	
Configuration	
Measurement Control	BT_Sig_Conf_BER_Search_Control
Loopback Parameters	BT_Sig_Conf_BER_Search_Settings
Fetch	BT_Sig_Fetch_BER_Search
Read	BT_Sig_Read_BER_Search
RCT General Configurations	

Input & Output	
Input Output	BT_Conf_IO
Reference Frequencies	BT_Conf_Ref_Frequencies
Netmode	
Signalling Mode	BT_Sig_Conf_Test_Mode
Utility	
RCT_Booked	BT_RCT_Booked
RCT_Vacated	BT_RCT_Vacated
RCT_Get_Sharing_State	BT_RCT_Get_Sharing_State
Bench Calibration	
Change Calibration Paths	BT_Util_Conf_Calibration
Get Calibration Offsets	BT_Util_GetCalibrationOffsets

7.3.4 CDMA Test Library

7.3.4.1 General

Name of the dynamic link library (DLL):	CDMA.DLL
Name of the help file (HLP):	CDMA.HLP
License required	R&S TS-LCDM
Supported devices:	Radio Communication Tester CMU200 with CDMAone signalling unit CMU-B81

The CDMA Test Library provides CDMA (Code Division Multiple Access) test functions for the function test and final test of mobile telephones. The Test Library contains functions for

- opening, configuring and closing the Radio Communication Tester present in the test system.
- carrying out measurements with the Radio Communication Tester present in the test system.

7.3.4.2 Entries in PHYSICAL.INI

Section [device->...]

Keyword	Value	Description
Type	String	*Mandatory entry* CMU
ResourceDesc	String	*Mandatory entry* VISA resource descriptor in the form: GPIB[board]::[primary Address]::[secundary Address]
ResourceDesc_RF_Nsig	String	VISA resource descriptor for the RF non-signaling part
ResourceDesc_CDMA_800Nsig	String	VISA resource descriptor for the CDMA 800 non-signaling part
ResourceDesc_CDMA_800Sig	String	VISA resource descriptor for the CDMA 800 signaling part
ResourceDesc_CDMA_1900Nsig	String	VISA resource descriptor for the CDMA 1900 non-signaling part
ResourceDesc_CDMA_1900Sig	String	VISA resource descriptor for the CDMA 1900 signaling part

Keyword	Value	Description
ResourceDesc_Audio	String	VISA resource descriptor for the audio part
Trace	0 / 1	*Optional entry* Enables/disables CMU driver-level tracing.
TraceFile	String	*Optional entry* Filename for CMU driver trace.

7.3.4.3 Entries in APPLICATION.INI

Section [bench->...]

Keyword	Value	Description
RadioComTester	String	*Mandatory entry* Reference to the device entry of the Radio Communication Tester in PHYSICAL.INI.
Simulation	0 / 1	*Optional entry* Blocks the simulation of the entered devices (value = 0). Enables simulation of the entered devices (value = 1). Default = 0
Trace	0 / 1	*Optional entry* Blocks the tracing function (value = 0), enables the tracing function (value = 1). Default = 0
ViewSharedMem-Blocks	0 / 1	*Optional entry* Enables/disables extended tracing. Default = 0

Optional entries

Entries for calibration.

Keyword	Value	Description
Calibration	0 / 1	Blocks the calibration function (value = 0), enables the calibration function (value = 1).
CalibrationFile<i>	String	Path and file name of the calibration files. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.

Keyword	Value	Description
CalibrationPath_ CDMA_Tx<i>	String	List of calibration tables for the CDMA-Tx path. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.
CalibrationPath_ CDMA_Rx<i>	String	List of calibration tables for the CDMA-Rx path. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.
CalibrationPath_ AUDIO_Ear<i>	String	List of calibration tables for the audio ear path. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.
CalibrationPath_ AUDIO_Mouth<i>	String	List of calibration tables for the audio mouth path. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.

7.3.4.4 Functions

```
Setup                           CDMA_Setup
Library Version                 CDMA_Lib_Version
Audio
   Configuration
      Generator                 CDMA_AF_Conf_Generator
      Switch AF Generator ON OFF    CDMA_AF_Conf_Gen_OnOff
      Analyzer                  CDMA_AF_Conf_Analyzer
   Measurements
      SingleTone                CDMA_AF_Meas_Voltage
Non Signalling
   Configuration
      RF Analyzer
         Settings               CDMA_NSig_Conf_RFA_Settings
         RF max expected power  CDMA_NSig_Conf_RFA_RF_Max_Level
         RF Channel             CDMA_NSig_Conf_RFA_Channel
         RF Frequency           CDMA_NSig_Conf_RFA_Frequency
      RF Generator
         Level                  CDMA_NSig_Conf_RFG_RF_Level
         Impairments            CDMA_NSig_Conf_RFG_Impairments
         Settings               CDMA_NSig_Conf_RFG_Settings
         RF Channel             CDMA_NSig_Conf_RFG_Channel
         RF Frequency           CDMA_NSig_Conf_RFG_Frequency
   Measurements
      Read
         Power/Modulation       CDMA_NSig_Read
```

Signalling	
Configuration	
BTS Parameters	
Global Settings	CDMA_Sig_Conf_BS_Settings
Level Settings	CDMA_Sig_Conf_BS_Levels
RF Channel	CDMA_Sig_Conf_RF_Channel
Traffic Channel	CDMA_Sig_Conf_TCH
PN Offset	CDMA_Sig_Conf_PN_Offset
Frame Offset	CDMA_Sig_Conf_Frame_Offset
Call Mode	CDMA_Sig_Conf_Call_Mode
MS Identity	
Network Parameters	CDMA_Sig_Conf_Netw_Param
Handoff Transition	
Handoff from CDMA	CDMA_Sig_Conf_Handoff_Cdma
Call Management	
Call Establishment	CDMA_Sig_Call
Release Call	CDMA_Sig_Call_Release
Registration	CDMA_Sig_Call_Registration
MS Identification	CDMA_Sig_Call_MS_ID
Call Status	CDMA_Sig_Call_Status
Execute Handoff	CDMA_Sig_HandOff
Measurements	
Custom measurements	CDMA_Sig_Custom_Measurements
Standby and Access Probe Power	
Read	CDMA_Sig_Read_SAP_Power
Overview	
Pilot Power	
Configuration	CDMA_Sig_Conf_Rx_Pilot_Power
Read	CDMA_Sig_Read_Rx_Pilot_Power
Rx Receiver Tests	
Receiver Quality	
Configurations	
Global Settings	CDMA_Sig_Conf_Rx_Global_Settings
Control	CDMA_Sig_Conf_Rx
Levels	CDMA_Sig_Conf_Rx_Lev
Impairments	CDMA_Sig_Conf_Rx_Impairments
Limits	CDMA_Sig_Conf_Rx_Lim
Read	CDMA_Sig_Read_Rx
Fetch	CDMA_Sig_Fetch_Rx
Tx Transmitter Tests	
Power	
Configurations	
Minimum Output	
Control	CDMA_Sig_Conf_Tx_Pow_Min
Levels / Limits	CDMA_Sig_Conf_Tx_Pow_Min_Lim
Maximum Output	
Control	CDMA_Sig_Conf_Tx_Pow_Max
Levels / Limits	CDMA_Sig_Conf_Tx_Pow_Max_Lim
Open Loop Time Response	
Levels	CDMA_Sig_Conf_Tx_Pow_OpenLoop
Gated Output	

```
                  Control            CDMA_Sig_Conf_Tx_Pow_Gated
                  Levels / Limits    CDMA_Sig_Conf_Tx_Pow_Gated_Lim
            Read
               Min/Max Power         CDMA_Sig_Read_Tx_Pow_MinMax
               Gated/OpenLoop Power  CDMA_Sig_Read_Tx_Pow_GatedOpenLoop
         Modulation
            Configurations
               Control               CDMA_Sig_Conf_Tx_Mod
               Levels                CDMA_Sig_Conf_Tx_Mod_Lev
               Impairments           CDMA_Sig_Conf_Tx_Mod_Impairments
               Limits                CDMA_Sig_Conf_Tx_Mod_Lim
            Read
               Read All              CDMA_Sig_Read_Tx_Mod_All
               Read Only One Meas    CDMA_Sig_Read_Tx_Mod
            Fetch
               Fetch Scal Over Mod   CDMA_Sig_Fetch_Tx_Mod
RCT General Configurations
   Network Selection                 CDMA_Conf_Network
   Test Mode                         CDMA_Conf_Test_Mode
   Input Output                      CDMA_Conf_IO
   Display On/Off                    CDMA_Conf_Display
Utility
   Calibration
      Change paths                   CDMA_Util_Conf_Calibration
      Get Calibration                CDMA_Util_Get_Calibration_Channel
   Clean Error Queue
      Set STB to 0                   CDMA_Util_CleanErrorQueue
Cleanup                              CDMA_Cleanup
```

7.3.5 CDMA2000 Test Library

7.3.5.1 General

Name of the dynamic link library (DLL):	CDMA2K.DLL
Name of the help file (HLP):	CDMA2K.HLP
License required	R&S TS-LC2K
Supported devices:	Radio Communication Tester CMU200 with CDMA2000 signalling unit CMU-B83

The CDMA2000 Test Library provides CDMAone and CDMA2000 (Code Division Multiple Access) test functions for the function test and final test of mobile telephones.

The Test Library contains functions for

- Network configuration
- RF generator / analyzer
- Call setup / release / handoff
- Power measurements
- Modulation measurements
- Receiver quality measurements
- Audio measurements

7.3.5.2 Entries in PHYSICAL.INI

Section [device->...]

Keyword	Value	Description
Type	String	*Mandatory entry* CMU
ResourceDesc	String	*Mandatory entry* VISA resource descriptor in the form: GPIB[board]::[primary Address]::[secondary Address]
ResourceDesc_ RF_NSig	String	VISA device properties and device description for the “RF non-signaling” component.
ResourceDesc_ CDMA2K_ 450MS_NSig	String	VISA device properties and device description for the “450 MHz non-signaling” component.

Keyword	Value	Description
ResourceDesc_ CDMA2K_ 450MS_Sig	String	VISA device properties and device description for the “450 MHz signaling” component.
ResourceDesc_ CDMA2K_ CELLMS_NSig	String	VISA device properties and device description for the “Cellular non-signaling” component.
ResourceDesc_ CDMA2K_ CELLMS_Sig	String	VISA device properties and device description for the “Cellular signaling” component.
ResourceDesc_ CDMA2K_ PCSMS_Nsig	String	VISA device properties and device description for the “PCS non-signaling” component.
ResourceDesc_ CDMA2K_ PCSMS_Sig	String	VISA device properties and device description for the “PCS signaling” component.
ResourceDesc_ CDMA2K_ IMT2KMS_NSig	String	VISA device properties and device description for the “IMT-2000 non-signaling” component.
ResourceDesc_ CDMA2K_ IMT2KMS_Sig	String	VISA device properties and device description for the “IMT-2000 signaling” component.
ResourceDesc_ Audio	String	VISA device properties and device description for the audio component.
Trace	0/1	*Optional entry* Blocks the tracing function of the driver (value = 0), enables the tracing function of the driver (value = 1). Default = 0
TraceFile	String	*Optional entry* Defines the path and the name of the trace file. Default = “”

7.3.5.3 Entries in APPLICATION.INI

Section [bench->...]

Keyword	Value	Description
RadioComTester	String	*Mandatory entry* Reference to the device entry of the Radio Communica-tion Tester in PHYSICAL.INI.
Simulation	0/1	*Optional entry* Blocks the simulation of the entered devices (value = 0). Enables simulation of the entered devices (value = 1). Default = 0
Trace	0/1	*Optional entry* Blocks the tracing function of the library (value = 0), enables the tracing function of the library (value = 1). Default = 0
ViewSharedMem-Blocks	0/1	*Optional entry* Enables/disables extended tracing. Default = 0
Calibration	0/1	*Optional entry* Blocks the calibration function (value = 0), enables the calibration function (value = 1). Default = 0
CalibrationFile<i>	String	Path and file name of the calibration files. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.
CalibrationPath_ CDMA2K_Tx<i>	String	List of calibration tables for the CDMA2K-Tx path. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.
CalibrationPath_ CDMA2K_Rx<i>	String	List of calibration tables for the CDMA2K-Rx path. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.

Keyword	Value	Description
CalibrationPath_ AUDIO_Ear	String	List of calibration tables for the audio ear path. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.
CalibrationPath_ AUDIO_Mouth	String	List of calibration tables for the audio mouth path. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.

7.3.5.4 Functions

```
Setup                                      CDMA2K_Setup
Library Version                            CDMA2K_Lib_Version
General Configurations
   Network Selection                       CDMA2K_ConfNetwork
   Test Mode                               CDMA2K_ConfTestMode
   Input Output                            CDMA2K_ConfIO
General Measurements (Sig&NSig)
   Configuration
      Burst configuration                  CDMA2K_BurstConfiguration
      Scalar or Array configuration        CDMA2K_ScalarArrayConfiguratio
n
   Custom Measurement                      CDMA2K_ConfCustom
   Fetch NPower                            CDMA2K_FetchNpower
   Read NPower                             CDMA2K_ReadNpower
Non Signalling
   Configuration
      RF Generator
         RF Levels                         CDMA2K_NSigConfRFGLevels
         Impairments                       CDMA2K_NSigConfRFGImpairments
         Settings                          CDMA2K_NSigConfRFGSettings
         RF Channel                        CDMA2K_NSigConfRFGChannel
         RF Frequency                      CDMA2K_NSigConfRFGFrequency
      RF Analyzer
         Settings                          CDMA2K_NSigConfRFASettings
         RF max expected power             CDMA2K_NSigConfRFAMaxLevel
         RF Channel                        CDMA2K_NSigConfRFAChannel
         RF Frequency                      CDMA2K_NSigConfRFAFrequency
         RF Frequency Offset               CDMA2K_NSigConfRFAFreqOffset
   Measurements
      Read modulation quality              CDMA2K_NSigReadMQuality
      Fetch modulation quality             CDMA2K_NSigFetchMQuality
Signalling
   Call Management
      Call Establishment                   CDMA2K_SigCall
      Release Call                         CDMA2K_SigCallRelease
      Execute Handoff                      CDMA2K_SigHandOff
      MS Identification?                    CDMA2K_SigCallMSID
      Call Status?                          CDMA2K_SigCallStatus
   Configuration
      Handoff
         HandoffTarget                      CDMA2K_SigConfHandoffTarget
      Measurement
         Level configuration                CDMA2K_SigMeasConfLev
         Code Domain Power levels           CDMA2K_SigMeasCDPowerConfLev
         Open loop time res conf            CDMA2K_SigMeasOpenLoopTimeResC
onf
         Receiver quality level conf        CDMA2K_SigReceiverQualMeasConf
Lev
         Receiver quality limit/rep conf    CDMA2K_SigReceiverQualConfLimi
tRep
         Channel quality power ctrl bit     CDMA2K_SigChanQualPowCtrlBit
         Impairment configuration           CDMA2K_SigImpairmentMeasConf
      Services
         Loop Configuration                 CDMA2K_LoopConfiguration
         Speech Configuration               CDMA2K_SpeechConfiguration
         TestDataService
Configuration      CDMA2K_TestDataServiceConfiguration
         Sub configuration for test data
              TestDataService FCH
Conf      CDMA2K_TestDataSerFCHConfiguration
              TestDataService SCH0 Conf
    CDMA2K_TestDataSerSCH0Configuration
      BS Signal
         Main Base Settings                 CDMA2K_SigConfBSSettings
         RF & Forward fund channel          CDMA2K_SigConfRFChanFFundChan
      Network
         System parameters                  CDMA2K_SigConfNetwSystemParam
         Network and mobile Identities      CDMA2K_SigConfNetwIdentities
         Paging options                     CDMA2K_SigConfNetwPaging
         Registration options               CDMA2K_SigConfNetwRegistration
   Measurements
      Overview
         Scalar
            Fetch overview                  CDMA2K_SigFetchOverOver
            Read overview                   CDMA2K_SigReadOverOver
            Fetch channel quality           CDMA2K_SigFetchOverCQuality
            Fetch channel quality extended  CDMA2K_SigFetchOverCQualityEx
            Read channel quality            CDMA2K_SigReadOverCQuality
            Read channel quality extended   CDMA2K_SigReadOverCQualityEx
      Power
         Scalar
            Fetch min/max power             CDMA2K_SigFetchMinMaxPower
            Read min/max power              CDMA2K_SigReadMinMaxPower
            Fetch gated output power        CDMA2K_SigFetchGatedOPower
            Read gated output power         CDMA2K_SigReadGatedOPower
            Fetch open loop time response   CDMA2K_SigFetchOLoopTimeResp
            Read open loop time response    CDMA2K_SigReadOLoopTimeResp
         Array
            Fetch array gated output power  CDMA2K_SigFetchGatedOPowerArra
y
            Read array gated output power   CDMA2K_SigReadGatedOPowerArray
            Fetch array open loop time resp CDMA2K_SigFetchOLoopTimeRespAr
ray
            Read array open loop time resp  CDMA2K_SigReadOLoopTimeRespArr
ay
      Standby and Access Probe Power
         Scalar
            Read SAP power                  CDMA2K_SigReadSAPower
      Code Domain Power
         Scalar
            Fetch code domain power         CDMA2K_SigFetchCDPower
            Read code domain power          CDMA2K_SigReadCDPower
         Array
            Fetch array code domain power   CDMA2K_SigFetchCDPowerArray
            Read array code domain power    CDMA2K_SigReadCDPowerArray
      Modulation
         Scalar
            Fetch overview modulation       CDMA2K_SigFetchOverMod
            Read overview modulation        CDMA2K_SigReadOverMod
            Fetch EVMag/MErr/PErr meas      CDMA2K_SigFetchMod
            Read EVMag/MErr/PErr meas       CDMA2K_SigReadMod
         Array
            Fetch Arr EVMag/MErr/PErr meas  CDMA2K_SigFetchModArray
            Read Arr EVMag/MErr/PErr meas   CDMA2K_SigReadModArray
      Receiver Quality
         Scalar
            Fetch RX quality                CDMA2K_SigFetchRXQuality
            Read RX quality                 CDMA2K_SigReadRXQuality
Audio
   Configuration
      Generator                             CDMA2K_AfConfGeneratorSettings
      Analyzer                              CDMA2K_AfConfAnalyzerSettings
      Switch AF Generator ON OFF            CDMA2K_AfConfGeneratorOnOff
      Speech Coder Options                  CDMA2K_AfConfSpeechCoder
      Multitone
         Set Tone Definition                CDMA2K_AfConfMultitoneToneDef
         Set Analyzer Levels                CDMA2K_AfConfMultitoneAnaLvl
         Set Limits                         CDMA2K_AfConfMultitoneLimits
   Measurements
      SingleTone                            CDMA2K_AfMeasVoltage
      Multitone
         Level Array                        CDMA2K_AfReadMultitoneLvlArray
         Matching Array                     CDMA2K_AfReadMultitoneMatchArr
ay
Utility
   Calibration
     Change path                            CDMA2K_UtilConfCalibration
     Get Calibration                        CDMA2K_UtilGetCalibrationChann
```

el	
Cleanup	CDMA2K_Cleanup

7.3.6 GSM Test Library

7.3.6.1 General

Name of the dynamic link library (DLL):	GSM.DLL
Name of the help file (HLP):	GSM.HLP
License required	R&S TS-LGSM for GSM R&S TS-LGSM + R&S TS-LGPR for GPRS
Supported devices:	Radio Communication Tester CMD55, CMU200

The GSM Test Library provides GSM test functions (Global System for Mobile Communication) for the function test and final test of mobile telephones. The Test Library contains functions for

- opening, configuring and closing the Radio Communication Tester present in the test system.
- carrying out measurements with the Radio Communication Tester present in the test system.

7.3.6.2 Entries in PHYSICAL.INI

Section [device->...]

Keyword	Value	Description
Type	String	*Mandatory entry* CMD55 or CMU
ResourceDesc	String	*Mandatory entry* VISA resource descriptor in the form: GPIB[card number]::[primary address]::[secondary address]
ResourceDesc_ RF_NSig	String	(CMU only) VISA resource descriptor for the „RF non-signaling“ component.
ResourceDesc_ GSM_850Nsig	String	(CMU only) VISA resource descriptor for the "GSM 850 non-signaling" component
ResourceDesc_ GSM_850Sig	String	(CMU only) VISA resource descriptor for the "GSM 850 signaling" component
ResourceDesc_ GSM_900Nsig	String	(CMU only) VISA resource descriptor for the „GSM 900 non-signaling“ component.

Keyword	Value	Description
ResourceDesc_ GSM_900Sig	String	(CMU only) VISA resource descriptor for the „GSM 900 signaling“ component.
ResourceDesc_ GSM_1800NSig	String	(CMU only) VISA resource descriptor for the „GSM 1800 non-signaling“ component.
ResourceDesc_ GSM_1800Sig	String	(CMU only) VISA resource descriptor for the „GSM 1800 signaling“ component.
ResourceDesc_ GSM_1900NSig	String	(CMU only) VISA resource descriptor for the „GSM 1900 non-signaling“ component.
ResourceDesc_ GSM_1900Sig	String	(CMU only) VISA resource descriptor for the „GSM 1900 signaling“ component.
ResourceDesc_Audio	String	(CMU only) VISA resource descriptor for the audio component.
Trace	0 / 1	*Optional entry* Blocks the tracing function of the driver (value = 0), enables the tracing function of the driver (value = 1). Default = 0
TraceFile	String	*Optional entry* Defines the path and the name of the trace file. Default = „“

7.3.6.3 Entries in APPLICATION.INI

Section [bench->...]

Keyword	Value	Description
RadioComTester	String	*Mandatory entry* Reference to the device entry of the Radio Communication Tester in PHYSICAL.INI.
Simulation	0 / 1	*Optional entry* Blocks the simulation of the entered devices (value = 0). Enables simulation of the entered devices (value = 1). Default = 0
Trace	0 / 1	*Optional entry* Blocks the tracing function of the library (value = 0), enables the tracing function of the library (value = 1). Default = 0

Keyword	Value	Description
ViewSharedMem-Blocks	0 / 1	*Optional entry* Enables/disables extended tracing. Default = 0

Optional entries

Entries for calibration.

Keyword	Value	Description
Calibration	0 / 1	Blocks the calibration function (value = 0), enables the calibration function (value = 1).
CalibrationFile<i>	String	Path and file name of the calibration files. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.
CalibrationPath_ GSM_Tx<i>	String	List of calibration tables for the GSM-Tx path. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.
CalibrationPath_ GSM_Rx<i>	String	List of calibration tables for the GSM-Rx path. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.
CalibrationPath_ AUDIO_Ear<i>	String	List of calibration tables for the audio ear path. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.
CalibrationPath_ AUDIO_Mouth<i>	String	List of calibration tables for the audio mouth path. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.

7.3.6.4 Functions

```
Setup                                  GSM_Setup
Library Version                        GSM_Lib_Version
CMU Version                            GSM_CMU_Version
AF
   Configuration
      Analyzer                         GSM_AF_Conf_Analyzer
      Analyzer Channel 2               GSM_AF_Conf_Analyzer2
      Analyzer level                   GSM_AF_Conf_Analyzer_Level
      Generator                        GSM_AF_Conf_Generator
      Generator Channel 2              GSM_AF_Conf_Generator2
      Generator/Voltmeter Path         GSM_AF_Conf_Gen_Volt_Paths
      Multitone: Set Tone Definition   GSM_AF_Conf_Multitone_ToneDef
      Multitone: Set Analyzer Levels   GSM_AF_Conf_Multitone_AnaLvl
      Multitone: Set limits            GSM_AF_Conf_Multitone_Limits
      Multitone/Frequency              GSM_AF_Conf_Multsin_Freq
      Multitone/Level                  GSM_AF_Conf_Multsin_Lvl
      Multitone/State                  GSM_AF_Conf_Multsin_State
      SLR/RLR                          GSM_AF_Conf_SLR_RLR
      Spectrum Analyzer Settings (Center/Span)
                                       GSM_AF_Conf_Spec_Analyzer_CS
      Spectrum Analyzer Settings (Start/Stop)
                                       GSM_AF_Conf_Spec_Analyzer_SS
      Switch AF Generator ON OFF       GSM_AF_Conf_Gen_OnOff
   Measure
      Custom measurement               GSM_AF_CustomMeasurement
      Fetch
         SingleTone (Fetch)            GSM_AF_Fetch_Singletone
         Multitone: Level Array (fetch)GSM_AF_Fetch_Multitone_Lvl_Array
      Read
         Multitone: Level Array        GSM_AF_Meas_Multitone_Lvl_Array
        Multitone: Matching Array     GSM_AF_Meas_Multitone_Match_Array
         Multitone/AllLevel            GSM_AF_Meas_Multitone
         ReceivingLoudnessRating       GSM_AF_Meas_RLR
         SendingLoudnessRating         GSM_AF_Meas_SLR
         SingleTone                    GSM_AF_Meas_Voltage
         Spectrum Analyzer Array       GSM_AF_Read_Spectrum
         Spectrum Analyzer Freq        GSM_AF_Read_Spectrum_Freq
Non Signalling
   Configuration
      Analyzer Frequency               GSM_NonSig_Conf_Analyzer_Freq
      Burst Analysis                   GSM_NonSig_Conf_Burst_Analysis
      Generator Frequency              GSM_NonSig_Conf_Generator_Freq
      Modulation                       GSM_NonSig_Conf_Mod
      Narrow-band power                GSM_NonSig_Conf_NPower
      Power                            GSM_NonSig_Conf_Pow
      Power and Modulation             GSM_NonSig_Conf_PowMod
      Spectrum analysis                GSM_NonSig_Conf_Spectrum
      Spectrum due to Modulation       GSM_NonSig_Conf_Spect_Mod
      Spectrum due to Switching        GSM_NonSig_Conf_Spect_Switch
      Switch RF Generator ON OFF       GSM_NonSig_Conf_RF_Gen_OnOff
```

Trigger	GSM_NonSig_Conf_Trigger
8PSK modulation only	
8PSK Error Vector Magnitude	GSM_NonSig_Conf_EVM
8PSK Magnitude Error	GSM_NonSig_Conf_MagnErr
8PSK Phase Error	GSM_NonSig_Conf_PhaseErr
8PSK Overview	GSM_NonSig_Conf_Overview_8PSK
8PSK Power	GSM_NonSig_Conf_Power_8PSK
Measure	
Burst Analysis	GSM_NonSig_Meas_Burst
Frequency Error	GSM_NonSig_Meas_Freq_Error
Narrow-band power	GSM_NonSig_Meas_NPower
Phase Error Peak	GSM_NonSig_Meas_Phase_Error_PK
Phase Error RMS	GSM_NonSig_Meas_Phase_Error_RMS
Power Burst Array	GSM_NonSig_Meas_PowBurstArray
Power Average	GSM_NonSig_Meas_Power_AVG
Power Peak	GSM_NonSig_Meas_Power_PK
Power Time	GSM_NonSig_Meas_Power_Time
Spectrum due to Modul. Array	GSM_NonSig_Meas_Spect_Mod_Array
Spectrum due to Switching Array	GSM_NonSig_Meas_Spect_Switch_Array
8PSK modulation only	
8PSK Error Vector Magnitude	GSM_NonSig_Meas_EVM
8PSK Magnitude Error	GSM_NonSig_Meas_MagnErr
8PSK Overview	GSM_NonSig_Meas_Overview_8PSK
8PSK Phase Error	GSM_NonSig_Meas_PhaseErr
8PSK Power	GSM_NonSig_Meas_Power_8PSK
Signalling	
Configuration	
BER Configuration	GSM_Sig_Conf_BER
BER power levels	GSM_Sig_Conf_BER_PowLev
BER sent frames	GSM_Sig_Conf_BER_SFrames
BS RF Parameters	GSM_Sig_Conf_BS_RF
BS Signal Options	GSM_Sig_Conf_BssMultislot
BS Simulation Parameters	GSM_Sig_Conf_BS_Simulation_Param
BS SpeechMode	GSM_Sig_Conf_BS_SpeechMode
Burst Analysis	GSM_Sig_Conf_Burst_Analysis
Change Mode GSM - GPRS	GSM_Sig_Conf_Mode_GPRS
Destination network (handover)	GSM_Sig_Conf_Handoff_Gsm
Location Update	GSM_Sig_Conf_Loc_Update
Modulation	GSM_Sig_Conf_Mod
Modulation XPER configuration	GSM_Sig_Conf_ModXPER
MS Signal Options	GSM_Sig_Conf_MssMultislot
Narrow-Band Power	GSM_Sig_Conf_NbPow
Network Handover	GSM_Sig_Conf_Network_Handover
Network Support	GSM_Sig_Conf_NetwSupport
Network Signaling Modes Configuration	GSM_Sig_Conf_NetworkSigModes
Network Timeouts Configuration	GSM_Sig_Conf_NetworkTimeouts
PCL, TCH and TS	GSM_Sig_Conf_PCL_TCH_TS
Power Base	GSM_Sig_Conf_Power_Base
Power	GSM_Sig_Conf_Pow
Power and Modulation	GSM_Sig_Conf_PowMod

```
      Power vs PCL configuration       GSM_Sig_Conf_PowvsPCL
      Spectrum analysis                GSM_Sig_Conf_Spectrum
      Spectrum due to Modulation       GSM_Sig_Conf_Spect_Mod
      Spectrum due to Switching        GSM_Sig_Conf_Spect_Switch
      Traffic Channel Speech Coding    GSM_Sig_Conf_TCHSpeechCoding
      Trigger                          GSM_Sig_Conf_Trigger
      GPRS/EGPRS
         Change GPRS Signalling Status / TBF Connection
                                       GSM_Sig_Conf_StatusGprs
         Bit Stream                    GSM_Sig_Conf_NetwBitStream
         BLER Configuration            GSM_Sig_Conf_BLER
         Coding Scheme                 GSM_Sig_Conf_CodingScheme
         Network Identity RAC          GSM_Sig_Conf_NetwRac
         Packed Data Service Options   GSM_Sig_Conf_NetwPdatService
         Puncturing Scheme             GSM_Sig_Conf_PuncturingScheme
      Multislot
         BS Signal Options             GSM_Sig_Conf_BssPdatService
         CEST Options                  GSM_Sig_Conf_CestMultislot
         MS Signal Options             GSM_Sig_Conf_MssPdatService
         Multislot Options             GSM_Sig_Conf_PdatMultiSlot
         Power Multislot Options       GSM_Sig_Conf_PowerMsl
      8PSK modulation only
         Config 8PSK modulation        GSM_Sig_Conf_8PSKModulation
         8PSK Error Vector Magnitude   GSM_Sig_Conf_EVM
         8PSK Magnitude Error          GSM_Sig_Conf_MagnErr
         8PSK Overview                 GSM_Sig_Conf_Overview_8PSK
         8PSK Phase Error              GSM_Sig_Conf_PhaseErr
         8PSK Power                    GSM_Sig_Conf_Power_8PSK
   Fetch
      Average power                    GSM_Sig_Fetch_Power_Avg
      BER                              GSM_Sig_Fetch_BER
      Fast BER                         GSM_Sig_Fetch_FBER
      Frequency error                  GSM_Sig_Fetch_Freq_Error
      Phase error Peak                 GSM_Sig_Fetch_Phase_Error_PK
      Phase error RMS                  GSM_Sig_Fetch_Phase_Error_RMS
      Power time template match        GSM_Sig_Fetch_Time_Template
      RBER                             GSM_Sig_Fetch_RBER
      Spect. due to Switching match    GSM_Sig_Fetch_Spect_Switch_Match
      Spect. due to Modulation match   GSM_Sig_Fetch_Spect_Mod_Match
   Handover
      Dual-Band HandOver               GSM_Sig_Handover
   Measure
      BER                              GSM_Sig_Meas_BER
      BLER - GPRS test mode BLER       GSM_Sig_Meas_BLER
      Burst Analysis                   GSM_Sig_Meas_Burst
      Custom measurement               GSM_Sig_Meas_Custom
      DBLER                            GSM_Sig_Meas_DBLER
      DBLER - GPRS Reduced Sig.        GSM_Sig_Meas_DBLER_GPRS
      Fast BER                         GSM_Sig_Meas_FBER
      Modulation XPER                  GSM_Sig_Meas_ModXPER
      Narrow-band Power                GSM_Sig_Meas_NbPow
```

Power Burst Array	GSM_Sig_Meas_PowBurstArray
Power Peak	GSM_Sig_Meas_Power_PK
Power Multislot Array	GSM_Sig_Meas_PowerMslArray
Power Multislot	GSM_Sig_Meas_PowerMsl
Power vs PCL	GSM_Sig_Meas_PowvsPCL
RBER	GSM_Sig_Meas_RBER
RSSI (RxLev and RxQual)	GSM_Sig_Meas_RxLev_RxQual
Spectrum due to Modul. Array	GSM_Sig_Meas_Spect_Mod_Array
Spectrum due to Modul. match	GSM_Sig_Meas_Spect_Mod_Match
Spectrum due to Switching Array	GSM_Sig_Meas_Spect_Switch_Array
Spectrum due to Switching match	GSM_Sig_Meas_Spect_Switch_Match
8PSK modulation only	
8PSK Error Vector Magnitude	GSM_Sig_Meas_EVM
8PSK Magnitude Error	GSM_Sig_Meas_MagnErr
8PSK Phase Error	GSM_Sig_Meas_PhaseErr
8PSK Overview	GSM_Sig_Meas_Overview_8PSK
8PSK Power	GSM_Sig_Meas_Power_8PSK
Registration, Call and Release	
Call Establishment	GSM_Sig_Call
Call Release	GSM_Sig_Release
Registration	GSM_Sig_WaitForMsRegist
Report	
Sense Call Status	GSM_Sig_Status_Call
Sense Multislot Class	GSM_Sig_MultislotClass
Sense Receiver Quality	GSM_Sig_Sens_RxQuality
Sense Reported Power Class	GSM_Sig_ReportedPcl
Report Internal Sig. State	GSM_Report_IntSigState
GPRS/EGPRS	
Sense GPRS Service Selection	GSM_Sig_Service_Gprs
Sense GPRS Signalling Status	GSM_Sig_Status_Gprs
General RCT Configurations	
Reset RCT	GSM_ResetRCT
Input & Output	
Input Output	GSM_Conf_In_Out
Reference Frequencies	GSM_Conf_Ref_Frequencies
Utility	
RCT_Booked	GSM_RCT_Booked
RCT_Vacated	GSM_RCT_Vacated
RCT_Get_Sharing_State	GSM_RCT_Get_Sharing_State
RF Generator	
RF Generator Settings	GSM_Conf_RF_Generator_Settings
RF Analyser	
Settings	GSM_Conf_RF_Analyzer
Trigger	GSM_Conf_RF_Trigger
RF NonSig Narrow-Band Power	
Configure narrow-band power	GSM_RF_Conf_NPower
Measure narrow-band power	GSM_RF_Meas_NPower
Test Mode	
Test Mode	GSM_Conf_Test_Mode
Network selection	
Network selection	GSM_Conf_Network

```
      Network selection with no sigON  GSM_Conf_Network_NoSigON
   Display
      Display Remote Report            GSM_Conf_DisplayRemoteReport
      Display                          GSM_Conf_Display
   Bench Calibration
      Change Calibration Paths         GSM_Util_Conf_Calibration
      Get Calibration Offsets (Chan)   GSM_Util_GetCalibrationOffsets
     Get Calibration Offsets (Freq)  GSM_Util_GetCalibrationOffsetsFreq
Cleanup                                GSM_Cleanup
```

7.3.7 RF Generator Test Library

7.3.7.1 General

Name of the dynamic link library (DLL):	RFGEN.DLL
Name of the help file (HLP):	RFGEN.HLP
License required	R&S TS-LBAS
Supported devices:	Radio Communication Tester CMU200 RF Signal Generator SME RF Signal Generator SMIQ

The RF Generator Test Library provides functions for the generation of a CW or modulated RF signal.

7.3.7.2 Entries in PHYSICAL.INI

Section [device->...]

Keyword	Value	Description

Type	String	*Mandatory entry*
		CMU = Radio Communication Tester CMU200
		SME = Signal Generator SME
		SMIQ = Signal Generator SMIQ

ResourceDesc	String	*Mandatory entry* VISA resource descriptor in the form: GPIB[board]::primaryAddress[::secondaryAddress]
ResourceDesc_ RF_Nsig	String	*Mandatory entry, CMU only* VISA resource descriptor for the RF non-signaling component
Trace	0 / 1	*Optional entry, CMU only* Blocks the tracing function of the driver (value = 0), enables the tracing function of the driver (value = 1). Default = 0
TraceFile	String	*Optional entry, CMU only* Defines the path and the name of the trace file. Default = “”

7.3.7.3 Entries in APPLICATION.INI

Section [bench->...]

Keyword	Value	Description
RFGenerator	String	*Mandatory entry* Reference to the device entry of the RF generator in PHYSICAL.INI.
Simulation	0 / 1	*Optional entry* Blocks the simulation of the entered devices (value = 0). Enables simulation of the entered devices (value = 1). Default = 0
Trace	0 / 1	*Optional entry* Blocks the tracing function of the library (value = 0), enables the tracing function of the library (value = 1). Default = 0
ViewSharedMem-Blocks	0 / 1	*Optional entry* Enables/disables extended tracing. Default = 0

Optional entries

Entries for calibration.

Keyword	Value	Description
Calibration	0 / 1	Blocks the calibration function (value = 0), enables the calibration function (value = 1).
CalibrationFile<i>	String	Path and file name of the calibration files. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.
CalibrationPath_ RF_Gen<i>	String	List of calibration tables for the RF Generator path. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.

7.3.7.4 Functions

```
Setup                             RFGEN_Setup
Library Version                   RFGEN_Lib_Version
Configuration
   Generator                      RFGEN_ConfGeneratorSettings
   Switch RF Generator ON OFF     RFGEN_ConfGeneratorOnOff
   Digital modulation             RFGEN_ConfDigitalModulation
Utility
   Calibration
      Change path                 RFGEN_UtilConfCalibration
      Get Calibration             RFGEN_UtilGetCalibrationOffset
Cleanup                           RFGEN_Cleanup
```

7.3.8 TDMA Test Library

7.3.8.1 General

Name of the dynamic link library (DLL):	TDMA.LIB
Name of the help file (HLP):	TDMA.HLP
License required	R&S TS-LTDM
Supported devices:	Radio Communication Tester CMU200

7.3.8.2 Entries in PHYSICAL.INI

Section [device->...]

Keyword	Value	Description
Type	String	*Mandatory entry* CMU
ResourceDesc	String	*Mandatory entry* VISA resource descriptor in the form GPIB[board]::primaryAddress[::secondaryAddress]
ResourceDesc_ RF_Nsig	String	VISA resource descriptor for the RF non-signaling part
ResourceDesc_ IS136_800Nsig	String	VISA resource descriptor for the IS136 800 non-signal-ing part
ResourceDesc_ IS136_800Sig	String	VISA resource descriptor for the IS136 800 signaling part
ResourceDesc_ IS136_1900Nsig	String	VISA resource descriptor for the IS136 1900 non-signal-ing part
ResourceDesc_ IS136_1900Sig	String	VISA resource descriptor for the IS136 1900 signaling part
ResourceDesc_Audi o	String	VISA resource descriptor for the audio part
Trace	0 / 1	*Optional entry* Enables/disables CMU driver-level tracing
TraceFile	String	*Optional entry* Filename for CMU driver trace

7.3.8.3 Entries in APPLICATION.INI

Section [bench->...]

Keyword	Value	Description
RadioComTester	String	*Mandatory entry* Reference to radio communication device entry
Simulation	0 / 1	*Optional entry* Enables/disables library-level simulation, default = 0
Trace	0 / 1	*Optional entry* Enables/disables tracing, default = 0
ViewSharedMem-Blocks	0 / 1	*Optional entry* Enables/disables extended tracing. Default = 0

Calibration entries (Optional entries)

Keyword	Value	Description
Calibration	0 / 1	Enables/disables calibration
CalibrationFile<i>	String	Path and file name of the calibration files. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.
CalibrationPath_TDMA_Tx<i>	String	List of calibration tables for the TDMA Tx path. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.
CalibrationPath_TDMA_Rx<i>	String	List of calibration tables for the TDMA Rx path. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.
CalibrationPath_AUDIO_Ear<i>	String	List of calibration tables for the audio ear path. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.
CalibrationPath_AUDIO_Mouth<i>	String	List of calibration tables for the audio mouth path. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.

7.3.8.4 Functions

Setup	TDMA_Setup
Library Version	TDMA_Lib_Version
Non Signalling	
RF Analyser Settings	TDMA_NSig_Conf_RF_Analyser
RF Generator Settings	TDMA_NSig_Conf_RF_Generator
RF Generator On/Off	TDMA_NSig_Conf_RF_Gen_OnOff
Custom measurements	TDMA_NSig_Custom_Measurements
Signalling	
Configuration	
BTS Parameters	
Global BTS Config	TDMA_Sig_Conf_BS_Settings
Digital traffic Channel (DTC)	TDMA_Sig_Conf_BS_DTC
DTC RF Level	TDMA_Sig_Conf_BS_DTC_RF_Level
DTC Slot	TDMA_Sig_Conf_BS_DTC_Slot
Speech Mode	TDMA_Sig_Conf_BS_SpeechMode
Network	
Network/ MS Identity	TDMA_Sig_Conf_Netw_Parm
Neighbour Channels	TDMA_Sig_Conf_Netw_NCH
DTC MAC	TDMA_Sig_Conf_Netw_DTC_MAC
Handoff Transition	
Handoff From TDMA	TDMA_Sig_Conf_Handoff_Tdma
Handoff Parameters (TDMA)	TDMA_Sig_Conf_Handoff_BS_Param
Call Established	
Digital traffic Channel (DTC)	TDMA_Sig_Conf_CEST_DTC
DTC RF Level	TDMA_Sig_Conf_CEST_DTC_RF_Level
DTC Slot	TDMA_Sig_Conf_CEST_DTC_Slot
Speech Mode	TDMA_Sig_Conf_CEST_SpeechMode
DTC MAC	TDMA_Sig_Conf_CEST_DTC_MAC
Call Management	
Call Establishment	TDMA_Sig_Call
Call Release	TDMA_Sig_Call_Release
Call Status	TDMA_Sig_Call_Status
MS Registration	TDMA_Sig_Call_Registration
MS Identification	TDMA_Sig_Call_MS_ID
Dialed Number	TDMA_Sig_Call_Dialed_Number
Execute Handoff	TDMA_Sig_Handoff
Custom measurements	TDMA_Sig_Custom_Measurements
Rx Receiver Tests	
Signalling (MAHO)	
Serving Cell	TDMA_Sig_Rx_MAHO_Serving
Neighbour Cells	TDMA_Sig_Rx_MAHO_Neighbour
Non Signalling	
Sensitivity	
Set Limits	TDMA_Conf_Rx_BER
Fetch	TDMA_Fetch_Rx_BER
Read	TDMA_Read_Rx_BER
Tx Transmitter Tests	
Power vs Time	
Configuration	
Measurement Control	TDMA_Conf_Tx_PVT

Set Limits	TDMA_Conf_Tx_PVT_Lim
Set Default Limits	TDMA_Conf_Tx_PVT_Def_Lim
Fetch	TDMA_Fetch_Tx_PVT
Read	TDMA_Read_Tx_PVT
Peak Power	
Read	TDMA_Read_Tx_Peak_Power
Adjacent Channel Power	
Configuration	
Measurement Control	TDMA_Conf_Tx_ACP
Set Limits	TDMA_Conf_Tx_ACP_Lim
Set Default Limits	TDMA_Conf_Tx_ACP_Def_Lim
Time Domain Specific	
Time	TDMA_Conf_Tx_ACP_Time
Fetch	
Fetch Frequency Domain	TDMA_Fetch_Tx_ACP_Freq
Fetch Time Domain	TDMA_Fetch_Tx_ACP_Time
Read	
Read Frequency Domain	TDMA_Read_Tx_ACP_Freq
Read Time Domain	TDMA_Read_Tx_ACP_Time
Modulation	
Configuration	
Measurement Control	TDMA_Conf_Tx_Mod
Set Limits	TDMA_Conf_Tx_Mod_Lim
Set Default Limits	TDMA_Conf_Tx_Mod_Def_Lim
Fetch	
Fetch All	TDMA_Fetch_Tx_Mod_All
Fetch Only One Meas	TDMA_Fetch_Tx_Mod
Read	
Read All	TDMA_Read_Tx_Mod_All
Read Only One Meas	TDMA_Read_Tx_Mod
RCT General Configurations	
Network Selection	TDMA_Conf_Network
Test Mode	TDMA_Conf_Test_Mode
Input Output	TDMA_Conf_IO
Reference Frequencies	TDMA_Conf_Ref_Frequencies
Utility	
RCT_Booked	TDMA_RCT_Booked
RCT_Vacated	TDMA_RCT_Vacated
RCT_Get_Sharing_State	TDMA_RCT_Get_Sharing_State
Change Calibration paths	TDMA_Util_Conf_Calibration
Cleanup	TDMA_Cleanup

7.3.9 WCDMA Test Library

7.3.9.1 General

Name of the dynamic link library (DLL):	WCDMA.DLL
Name of the help file (HLP):	WCDMA.HLP
License required	R&S TS-LWCD
Supported devices:	Radio Communication Tester CMU200

The WCDMA Test Library provides WCDMA test functions for the function test and final test of mobile telephones.

The Test Library contains functions for

- Network configuration
- Call setup / release / handoff
- Power measurements
- Modulation measurements
- Spectrum measurements
- Receiver quality measurements
- Audio measurements

7.3.9.2 Entries in PHYSICAL.INI

Section [device->...]

Keyword	Value	Description
Type	String	*Mandatory entry* CMU
ResourceDesc	String	*Mandatory entry* VISA resource descriptor in the form: GPIB[board]::[primary Address]::[secundary Address]
ResourceDesc_Audio	String	VISA resource descriptor for the Audio part
ResourceDesc_WCDMA_1900FDDMS_SIG	String	VISA resource descriptor for the WCDMA 1900 signaling part
ResourceDesc_WCDMA_1900FDDMS_NSIG	String	VISA resource descriptor for the WCDMA 1900 non-signaling part

Keyword	Value	Description
Trace	0 / 1	*Optional entry* Enables/disables CMU driver-level tracing.
TraceFile	String	*Optional entry* Filename for CMU driver trace.

7.3.9.3 Entries in APPLICATION.INI

Section [bench->...]

Keyword	Value	Description
RadioComTester	String	*Mandatory entry* Reference to the device entry of the Radio Communication Tester in PHYSICAL.INI.
Simulation	0 / 1	*Optional entry* Blocks the simulation of the entered devices (value = 0). Enables simulation of the entered devices (value = 1). Default = 0
Trace	0 / 1	*Optional entry* Blocks the tracing function (value = 0), enables the tracing function (value = 1). Default = 0
ViewSharedMem-Blocks	0 / 1	*Optional entry* Enables/disables extended tracing. Default = 0

Optional entries

Entries for calibration.

Keyword	Value	Description
Calibration	0 / 1	Blocks the calibration function (value = 0), enables the calibration function (value = 1).
CalibrationFile<i>	String	Path and file name of the calibration files. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.

Keyword	Value	Description
CalibrationPath_ WCDMA_Tx<i>	String	List of calibration tables for the WCDMA-Tx path. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.
CalibrationPath_ WCDMA_Rx<i>	String	List of calibration tables for the WCDMA-Rx path. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.
CalibrationPath_ AUDIO_Ear<i>	String	List of calibration tables for the audio ear path. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.
CalibrationPath_ AUDIO_Mouth<i>	String	List of calibration tables for the audio mouth path. <i> stands for a number from 1,2,3,...,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in the case it is 1.

7.3.9.4 Functions

```
Setup                                   WCDMA_Setup
Library Version                         WCDMA_Lib_Version
Configuration group (Sig&NSig)
   Network Selection                    WCDMA_ConfNetwork
   Test Mode                            WCDMA_ConfTestMode
   Uplink configuration                 WCDMA_ConfUplinkSettings
   Downlink configuration               WCDMA_ConfDownlinkSettings
   Downlink configuration CEST          WCDMA_ConfDownlinkSettingsCest
   Output power configuration           WCDMA_ConfOutputPowerSettings
   Channel settings                     WCDMA_ConfChannel
   Uplink Frequency                     WCDMA_ConfUplinkFrequency
   Downlink Frequency                   WCDMA_ConfDownlinkFrequency
   Downlink Physical Channel            WCDMA_ConfDownlinkPhysicalChan
   3GPP DL Ref channel conf             WCDMA_Conf3GPPRefChan
   TPC settings                         WCDMA_ConfTPC
   TPC repeat pattern (single)          WCDMA_TPCRepeatSinglePattern
   Trigger settings                     WCDMA_ConfTrigger
   Input Output                         WCDMA_ConfIO
   Signalling specific
      Handoff
         HandoffTarget                  WCDMA_SigConfHandoffTarget
      BS signal
         Dedicated channel (DCH)        WCDMA_ConfDedicatedChannel
         Individual physical channels   WCDMA_ConfPhysChannels
      Network
```

Network identity	WCDMA_SigConfNetwIdentity
Random access setting	WCDMA_SigConfRandomAccessSetting
Open loop power control	WCDMA_SigConfOpenLoopPowerControl
Non Signalling specific	
Generator state ON/OFF	WCDMA_NSigConfGeneratorState
UE Code	
DPDCH settings	WCDMA_NSigConfDPDCH
Generator	
DL physical channel conf	WCDMA_NSigConfRFGPhysRefChan
Call Management group (Sig)	
Call Establishment	WCDMA_SigCall
Release Call	WCDMA_SigCallRelease
Execute Handoff	WCDMA_SigHandOff
MS Identification?	WCDMA_SigCallMSID
Call Status?	WCDMA_SigCallStatus
Measurements group (Sig&NSig)	
Configuration	
Burst configuration	WCDMA_BurstConfiguration
Scalar or Array configuration	WCDMA_ScalarArrayConfiguration
Specific configuration	
Inner loop TPC configuration	WCDMA_ConfInnerLoopTpcMeas
Measurement length config	WCDMA_ConfMeasLength
Power vs slot table config.	WCDMA_ConfPowerVsSlotTable
Rx quality conf	WCDMA_ConfRxQuality
Custom Measurement	WCDMA_ConfCustom
Power	
Scalar	
Fetch min/max/off power	WCDMA_FetchMinMaxOffPower
Fetch inner loop TPC	WCDMA_FetchInnerLoopTPC
Fetch power vs slot table	WCDMA_FetchPowVsSlotTable
Fetch open loop power	WCDMA_FetchOLP
Array	
Fetch array inner loop TPC	WCDMA_FetchInnerLoopTPCArray
Code Domain Power	
Scalar	
Fetch CDP / Code manual	WCDMA_FetchCDPManual
Fetch CDP / Code automatic	WCDMA_FetchCDPAutomatic
Array	
Fetch array CDP / Code manual	WCDMA_FetchCDPManualArray
Fetch array CDP / Code auto	WCDMA_FetchCDPAutoArray
Modulation	
Scalar	
Fetch overview modulation	WCDMA_FetchOverMod
Fetch EVMag/MErr/PErr meas	WCDMA_FetchMod
Array	
Fetch Arr EVMag/MErr/PErr meas	WCDMA_FetchModArray
Spectrum	
Scalar	
Fetch ACLR Filter	WCDMA_FetchACLRfilter
Fetch ACLR FFT / OBW	WCDMA_FetchACLR_FFT_OBW
Fetch Emission Mask	WCDMA_FetchEmissionMask

```
      Array
         Fetch array ACLR FFT / OBW       WCDMA_FetchACLR_FFT_OBW_Array
         Fetch array Emission Mask        WCDMA_FetchEmissionMaskArray
   Rx quality
      Fetch Rx quality                    WCDMA_FetchRxQuality
Audio
   Configuration
      Generator                           WCDMA_AfConfGeneratorSettings
      Analyzer                            WCDMA_AfConfAnalyzerSettings
      Switch AF Generator ON OFF          WCDMA_AfConfGeneratorOnOff
      Speech Coder Options                WCDMA_AfConfSpeechCoder
      Multitone
         Set Tone Definition              WCDMA_AfConfMultitoneToneDef
         Set Analyzer Levels              WCDMA_AfConfMultitoneAnaLvl
         Set Limits                       WCDMA_AfConfMultitoneLimits
   Measurements
      SingleTone                          WCDMA_AfMeasVoltage
      Multitone
         Level Array                      WCDMA_AfReadMultitoneLvlArray
         Matching Array                   WCDMA_AfReadMultitoneMatchArray
Utility
   Calibration
      Change path                         WCDMA_UtilConfCalibration
      Get Calibration                     WCDMA_UtilGetCalibrationChannel
Cleanup                                   WCDMA_Cleanup
```

7.4 Radio Communication Support Libraries

7.4.1 Final Test Fixture Library

7.4.1.1 General

Name of the dynamic link library (DLL):	FINFIX.DLL
Name of the help file (HLP):	FINFIX.HLP
License required	R&S TS-LBAS
Supported devices:	Fixture interface for USB and R&S TS-PRL1

The Final Test Fixture Library provides functions to control the Fixture Interface for USB and R&S TS-PRL1:

- Open Collector Outputs
- Logical Inputs
- Relay Contacts for general use
- Audio Amplifier

7.4.1.2 Entries in PHYSICAL.INI

Section [device->...]

Keyword	Value	Description

Type	String	*Mandatory entry*
		PRL1 = R&S TS-PRL1 interface
		MEMPIO = USB interface

ResourceDesc	String	*Mandatory entry* VISA device properties and device description in the form: PXI[segment number]::[device number]::[function]:: INSTR for R&S TS-PRL1 SERIAL::[serial number] for USB interface

Keyword	Value	Description
DigitalInputs	4 / 8	*Optional Entry* Defines the number of digital inputs (4 or 8). This value depends on the jumper settings of JP1: 4 = JP1 not fitted 8 = JP1 fitted Default = 4

NOTE:

These entries may also appear in the APPLICATION.INI. In this case, the values from APPLICATION.INI override the values from PHYSICAL.INI. This is useful if different applications require fixtures with different serial numbers and/or digital inputs.

7.4.1.3 Entries in APPLICATION.INI

Section [bench->...]

Keyword	Value	Description
FinalFixture	String	*Mandatory entry* Reference to the device entry of the final test fixture in PHYSICAL.INI.
Simulation	0/1	*Optional entry* Blocks the simulation of the entered devices (value = 0). Enables simulation of the entered devices (value = 1). Default = 0
Trace	0/1	*Optional entry* Blocks the tracing function of the library (value = 0), enables the tracing function of the library (value = 1). Default = 0

7.4.1.4 Functions

```
Setup                              FINFIX_Setup
Lib version                        FINFIX_Lib_Version
Set all                            FINFIX_SetAll
Set open collector                 FINFIX_SetOpenCollector
Set audio                          FINFIX_SetAudio
Set relay                          FINFIX_SetRelay
Get all                            FINFIX_GetAll
Get input                          FINFIX_GetInput
Cleanup                            FINFIX_Cleanup
```

7.4.2 Path Characterization Library

7.4.2.1 General

Name of the dynamic link library (DLL):	PATHCHAR.DLL
Name of the help file (HLP):	PATHCHAR.HLP
License required	R&S TS-LBAS
Supported devices:	Source Instruments: – CMU200 – CMD55 – SME – SMIQ Measurement Instruments – CMU200 – CMD55 – NRVD – NRVS

The path characterization library provides functions used to measure signal path characteristics in a test system using the R&S GTSL software.

To use the routines of this library RESMGR_Setup from the R&S GTSL resource manager must be executed first. This function loads the physical and application layer INI files.

7.4.2.2 Entries in PHYSICAL.INI

No entries.

7.4.2.3 Entries in APPLICATION.INI

Section [bench->...]

Keyword	Value	Description
Trace	0 / 1	*Optional entry* 0 : Disable tracing 1 : Enable tracing Default: 0

Keyword	Value	Description
Simulation	0 / 1	*Optional entry* 0 : Disable simulation mode 1 : Enable simulation mode Default: 0 When enabled no source and measurement instrument are required to use the library. This is useful for testing.
CalibrationFile	String	Path and filename of the first calibration file
CalibrationFile<i>	String	Path and filename of the second, third... calibration file, where <i> is a number from 2 ... n

7.4.2.4 Entries in CALIBRATION.CAL

The calibration files also have the INI style (see also chapter 5.1 Syntax). On the one hand these files contain the configuration information for the whole characterization process. On the other hand the result of this process, the calibration tables, are also stored in these files.

Section [PathInfo->Name]

There is a section for every signal path to be measured. If the characteristics of a path in the system is determined with the reference method, two sections must be defined. The name of such a section must have the format [PathInfo->XY]:

PathInfo:	Keyword for path information sections
XY:	Arbitrary name for a path

Keywords used by the library:

Keyword	Value	Description
Perform	0 / 1	*Optional entry* 0 : Do not perform measurements for that path 1 : Perform measurements Default: 1 This flag can be read with the function `PATHCHAR_PathGetPerformFlag`. If this flag is zero the sequence should skip the characterization of that path.

Keyword	Value	Description
Normalization	0 / 1	*Optional entry* 0 : Do not perform a normalization measurement 1 : Path is configured for a normalization measurement. Default: 0 This flag can be read with the function `PATHCHAR_PathGetNormalizationFlag`. If set a reference (or normalization) measurement should be performed and the result is normally stored in a reference buffer. The contents of the reference buffer is used to calculate the correction offset when the path characteristics is measured.
Scan[X]	String	*Mandatory entry* Specifies the frequency ranges (scan) for which the characteristics of a path should be measured. The value of this key is a section within this file. X is a counter and is optional for the first scan in a path information section. The second key for a scan must have the number 2.

Examples:

```
[PathInfo->NormalizationGsmTx]
Perform = 1
Normalization = 1
Scan1 = ScanGsmTx1
Scan2 = ScanGsmTx2

[PathInfo->GsmTx]
Perform = 1
Scan1 = ScanGsmTx1
Scan2 = ScanGsmTx2

[PathInfo->GsmRx]
Perform = 1
Scan = ScanGsmRx
```

Scan Section [ScanName]

There is a section for every scan (frequency range). The name of such a section is arbitrary. It describes the instruments with it's settings used for the characterization process. Furthermore it holds the calibration table with the basic attenuation value. The table values are stored in the format „frequency = offset“. Note that the key „frequency“ is a floating point value representing the frequency in Hz! The following further keys are used by the library:

Keyword	Value	Description
Description	String	*Optional entry* Arbitrary text
SourceInst	String	*Optional entry* Specifies the source instrument. The value for that key is a device section in the physical layer INI file. The following device types are supported: CMU CMD55 SMIQ SME If you use a ”Golden Device” as source instrument you can omit this entry
SourceLevel	Floating Point	*Optional entry* Specifies the level for the source instrument in dBm. Have a look to the description of the function PATHCHAR_SourceInstSetup for details. Default: 0.0
SourcePort	Integer	*Optional entry* Specifies the port of the source instrument. Have a look to the description of the function PATHCHAR_SourceInstSetup for details. Default: 1
MeasurementInst	String	Specifies the measurement instrument. The value for that key is a device section in the physical layer INI file. The following device types are supported: CMU CMD55 NRVS NRVD If you use a „Golden Device“ as source instrument you can omit this entry.
MeasurementInst-Port	Integer	*Optional entry* Specifies the port of the measurement instrument. Have a look to the description of the function PATHCHAR_MeasInstSetup for details. Default: 1

Keyword	Value	Description
BasicAttenuation	Floating Point	*Optional entry* The correction value (correction_value) for a specific frequency is calculated from the basic attenuation (basic_attenuation) and the frequency dependent stored offset in the table (stored_offset): correction_value = basic_attenuation + stored_offset Default: 0.0

Example:

```
[ScanGsmTx1]
Description = This is a description of scan GsmTx1
SourceInst = device->CMU
SourceLevel = -10.0
SourcePort = 2
MeasurementInst = device->CMU
MeasurementPort = 4
BasicAttenuation = 0.0
890.0E6 = "0.00"
895.0E6 = "0.00"
900.0E6 = "0.00"
905.0E6 = "0.00"
910.0E6 = "0.00"
915.0E6 = "0.00"
920.0E6 = "0.00"
925.0E6 = "0.00"
930.0E6 = "0.00"
935.0E6 = "0.00"
940.0E6 = "0.00"
945.5E6 = "0.00"
950.5E6 = "0.00"
955.5E6 = "0.00"
960.5E6 = "0.00"
```

7.4.2.5 Functions

Management	
Setup	PATHCHAR_Setup
Library Version	PATHCHAR_Lib_Version
Cleanup	PATHCHAR_Cleanup
File Functions	
Get First File Name	PATHCHAR_FileGetFirstName
Get Next File Name	PATHCHAR_FileGetNextName
Select File	PATHCHAR_FileSelect
Path Functions	
Get First Path Name	PATHCHAR_PathGetFirstName
Get Next Path Name	PATHCHAR_PathGetNextName
Select Path	PATHCHAR_PathSelect

Get Perform Flag	PATHCHAR_PathGetPerformFlag
Get Normalization Flag	PATHCHAR_PathGetNormalizationFlag
Scan Functions	
Get First Scan Name	PATHCHAR_ScanGetFirstName
Get Next Scan Name	PATHCHAR_ScanGetNextName
Select Scan	PATHCHAR_ScanSelect
Set Basic Attenuation	PATHCHAR_ScanSetBasicAttenuation
Calc Basic Attenuation	PATHCHAR_ScanCalcBasicAttenuation
Get Source Level	PATHCHAR_ScanGetSourceLevel
Save Scan	PATHCHAR_ScanSave
Show Scan	PATHCHAR_ScanShow
Step Functions	
Get First Step Frequency	PATHCHAR_StepGetFirstFreq
Get Next Step Frequency	PATHCHAR_StepGetNextFreq
Select Step	PATHCHAR_StepSelect
Set Correction Value	PATHCHAR_StepSetCorrectionValue
Set Reference Buffer Value	PATHCHAR_StepSetRefBufferValue
Get Reference Buffer Value	PATHCHAR_StepGetRefBufferValue
Reference Buffer	
Clear Buffer	PATHCHAR_RefBufferClear
Show Buffer	PATHCHAR_RefBufferShow
Source Instrument	
Setup Source Instrument	PATHCHAR_SourceInstSetup
Set Source Instrument Frequency	PATHCHAR_SourceInstSetFreq
Reset Source Instrument	PATHCHAR_SourceInstReset
Measurement Instrument	
Setup Measurement Instrument	PATHCHAR_MeasInstSetup
Set Measurement Instrument Frequency	PATHCHAR_MeasInstSetFreq
Read Measurement Value	PATHCHAR_MeasInstRead
Miscellaneous	
Get Instrument Handle	PATHCHAR_InstrumentGetHandle
Get Instrument Subhandle	PATHCHAR_InstrumentGetSubHandle

7.4.3 Power Supply Test Library

7.4.3.1 General

Name of the dynamic link library (DLL):	PSU.DLL
Name of the help file (HLP):	PSU.HLP
License required	R&S TS-LBAS
Supported devices:	Keithley 2306 Battery Charger / Simulator, Rohde & Schwarz NGMO2

The Power Supply Test Library contains functions for

- opening, configuring and closing the power supplies present in the test system.
- simulating the supply from the Power Supplies,
- simulating external chargers via the Power Supplies.
- measuring the charging current (operating current).

7.4.3.2 Entries in PHYSICAL.INI

Section [device->...]

Keyword	Value	Description
Type	String	*Mandatory entry* KI2306 or NGMO2
ResourceDesc	String	*Mandatory entry* VISA device properties and device description in the form: GPIB[card number]::[primary address]::[secundary ad-dress]

7.4.3.3 Entries in APPLICATION.INI

Section [bench->...]

Keyword	Value	Description
PowerSupply	String	*Mandatory entry* Reference to the device entry of the power supply in PHYSICAL.INI.

Keyword	Value	Description
Simulation	0 / 1	*Optional entry* Blocks the simulation of the entered devices (value = 0). Enables simulation of the entered devices (value = 1). Default = 0
Trace	0 / 1	*Optional entry* Blocks the tracing function (value = 0), enables the tracing function (value = 1). Default = 0

7.4.3.4 Functions

Setup	PSU_Setup
Library Version	PSU_Lib_Version
Configuration	
Set Limits	PSU_Conf_Limits
Set Voltage Output	PSU_Conf_Voltage
Set Source Parameters	PSU_Conf_Source
Set Output Bandwidth	PSU_Conf_Bandwidth
Set Output Impedance	PSU_Conf_Impedance
Set Current Range	PSU_Conf_Current_Range
Set Measurement Parameters	PSU_Conf_Measurement
Output	
Connect / Disconnect Output	PSU_Output_Mode
Measurement	
DC Current	PSU_Meas_Current_DC
DC Current with NPLC	PSU_Meas_Current_DC_NPLC
Pulse Current	PSU_Meas_Current_Burst
Pulse Current long integration	PSU_Meas_Current_LINT
Voltage	PSU_Meas_Voltage
Voltage at External DVM Input	PSU_Meas_Voltage_Ext
Cleanup	PSU_Cleanup

7.4.4 UUT Test Library

7.4.4.1 General

Name of the dynamic link library (DLL):	UUTLIB.DLL
Name of the help file (HLP):	UUTLIB.HLP
License required	R&S TS-LBAS
Supported devices:	User-specific mobile telephone with bottom connector

The UUT Test Library provides test functions for the function test and final test of mobile telephones. The Test Library contains functions especially adapted to the mobile telephone being tested. For every type of mobile phone, it is necessary to create a separate UUT Test Library, or adapt the existing example.

7.4.4.2 Entries in PHYSICAL.INI

Section [device->...]

Keyword	Value	Description
Type	String	*Mandatory entry* USER = user-defined DEMO = demo mode with popup dialog boxes GENERIC_SERIAL = initializes the serial interface and implements the generic read/write functions. any other user-defined entry
ResourceDesc	String	*Mandatory entry* User-specific device properties and device description, e.g.: COM2 for serial interface
...	Any	all further user-specific entries

The following keys are valid for the UUT type GENERIC_SERIAL and may be given in the physical or the application INI file:

Section [device->...]

Key	Value	Description
BaudRate	Number	*Optional entry* Baud rate, default = 19200
DataBits	5, 6, 7 or 8	*Optional entry* Number of data bits, default = 8
StopBits	1 or 2	*Optional entry* Number of stop bits, default = 1
Parity	String	*Optional entry* Parity setting. Valid entries are NONE, EVEN and ODD, default = NONE
Handshake	String	*Optional entry* Hardware/Software handshake settings. Valid entries are NONE, XON/XOFF, CTS/RTS and CTS/RTS/DTR. Default = NONE.
ReadTerminator	Number	*Optional entry* Terminator character in read operations as a decimal number between 0 and 255. Default = 13 (carriage return)

7.4.4.3 Entries in APPLICATION.INI

Section [bench->...]

Keyword	Value	Description
UUT	String	*Mandatory entry* Reference to the device entry of the UUT in PHYSICAL.INI.
Simulation	0 / 1	*Optional entry* Blocks the simulation of the entered devices (value = 0). Enables simulation of the entered devices (value = 1). Default = 0
Trace	0 / 1	*Optional entry* Blocks the tracing function (value = 0), enables the tracing function (value = 1). Default = 0
...	Any	all further user-specific entries

7.4.4.4 Functions

```
Setup                               UUTLIB_Setup
Library Version                     UUTLIB_Lib_Version
Set Resource ID                     UUTLIB_Set_Resource_ID
Mode Functions
   Network selection                UUTLIB_Mode_Network
   Set Test Mode                    UUTLIB_Exit_Mode_Test
   Exit Test Mode                   UUTLIB_Exit_Mode_Test
Communication Functions
   Start Call from Mobile           UUTLIB_Call_Start
   End Call from Mobile             UUTLIB_Call_End
   Accept Call from Test Set        UUTLIB_Call_Accept
Data Functions
   Get RSS from Mobile              UUTLIB_Data_GetRSS
   Continuous Transmission          UUTLIB_Data_StartContTx
   End Continuous Transmission      UUTLIB_Data_EndContTx
Store Functions
   <Create Class or Function Panel Window>
Audio Functions
   Audio control                    UUTLIB_Audio
Serial Communication Functions
   Send Command/Data to the Mobile  UUTLIB_Comm_Send
   Receive Data from the Mobile     UUTLIB_Comm_Receive
GTSL-Libs Calling Functions
   Network selection (GSM and UUT)  UUTLIB_Mode_Network_GSM
Cleanup                             UUTLIB_Cleanup
```

8 Signal Routing

This chapter describes switching of measurement signals in R&S CompactTSVP / R&S PowerTSVP systems making use of the Signal Routing Library.

8.1 R&S GTSL software for switched connections

There are three libraries in R&S GTSL that can make switched connections:

- Signal Routing Library ROUTE.DLL
- Switch Manager Library SWMGR.DLL
- The library for In-Circuit Test ICT.DLL (R&S EGTSL)

All three libraries can simultaneously administer a large number of measurement, stimulus, and switch modules. These different modules appear together in the test program as a large switch panel.

8.1.1 Signal Routing Library

The Signal Routing Library makes it possible to set up complex switched connections by means of switching commands. Switched connections can be automatically routed by the analog measurement bus, i.e. the software searches for free analog measurement bus lines and automatically switches the relays in the switching path.

Extensive switched connections can also be saved under a user-specific name and then called in the test program.

A R&S TS-LSRL software license is required for the Signal Routing Library.

NOTE:

The Signal Routing Library cannot be used together with the Switch Manager.

8.1.2 Switch Manager Library

The Switch Manager Library is the predecessor of the Signal Routing Library. It is a useful tool when compatibility is required with earlier applications created with the Switch Manager Library. Unlike the Signal Routing Library, the Switch Manager has no built-in “intelligence” and is not capable of routing switching paths automatically.

The Switch Manager is already included in the basic license for R&S TS-LBAS.

NOTE:

The Switch Manager cannot be used together with the Signal Routing Library.

8.1.3 ICT Library / R&S EGTSL

The In-Circuit Test Library and R&S EGTSL user interface (R&S EGTSL IDE) make internal connections for the In-Circuit Test. They use the same entries in the configuration files to do this as the two other libraries. However it is not possible to use R&S EGTSL for general connection tasks, such as in the functional test.

8.2 Analog measurement bus concept

The R&S CompactTSVP and R&S PowerTSVP systems allow for use of a large number of measurement and stimulus modules. These modules can be connected with the UUT (unit under test) either directly or through switch modules.

The *analog measurement bus* of the R&S CompactTSVP and R&S PowerTSVP systems connects measurement, stimulus, and switch modules with each other. The analog measurement bus offers eight lines, which are available on all slots. In this manner, UUT signals can be flexibly connected with the measurement and stimulus modules through the switch modules.

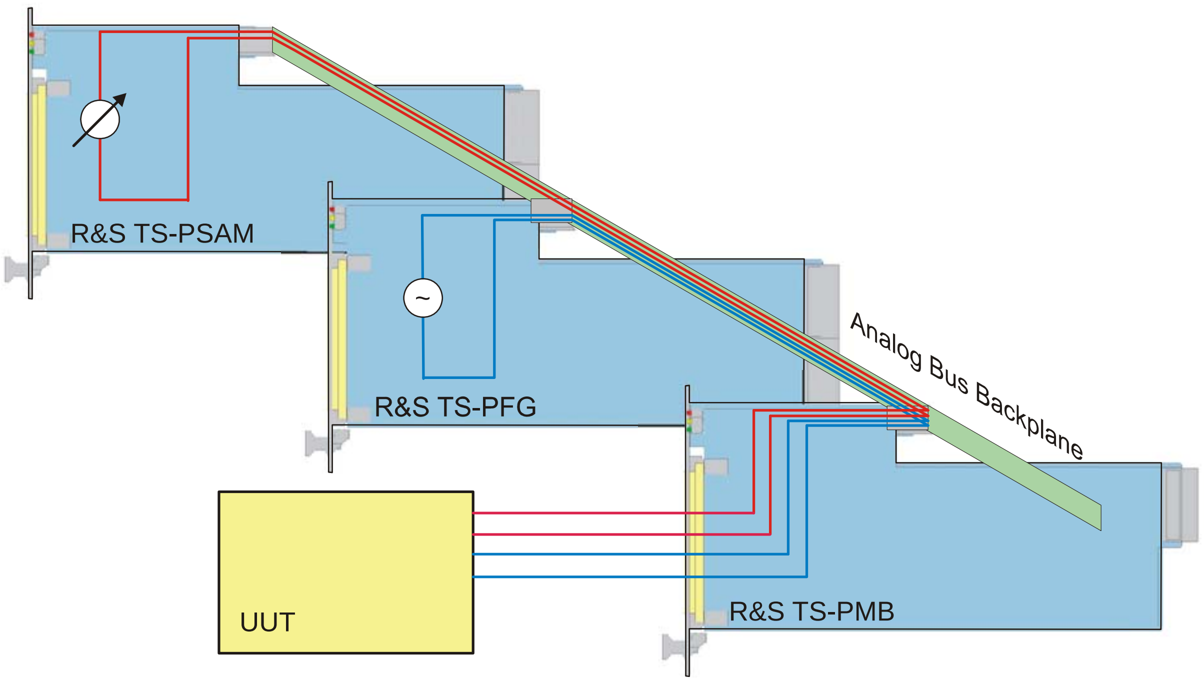


Figure 8-1 Analog measurement bus concept

The following table shows an overview of modules of the R&S CompactTSVP product line with switching elements:

Module name	Module type	Analog measurement bus access	Local multiplexers	Special features
R&S TS-PAM	Measurement module	x	x	
R&S TS-PDFT	Measurement/ stimulus module		x	Digital test module
R&S TS-PFG	Stimulus module	x		
R&S TS-PIO2	Measurement/ stimulus module	x	x	
R&S TS-PMB	Switch module	x		
R&S TS-PSAM	Measurement module	x	x	
R&S TS-PSM1	Switch module	x		Power switching module
R&S TS-PSM2	Switch module	x		Power switching module
R&S TS-PSU R&S TS-PSU12	Stimulus module	x	x	Power module
R&S TS-PSYS1/2	System module			

Table 8-1 Modules of the R&S CompactTSVP product line

With the exception of modules R&S TS-PDFT and R&S TS-PSYS, all modules provide access to the analog measurement bus. Some modules provide *local multiplexers*. If only a few signals need to be switched to test a UUT, the multiplexer in the module is frequently sufficient. If numerous channels are involved, however, multiplexing is performed by the analog measurement bus and switching modules.

The analog measurement bus consists of eight lines with identical capabilities, ABa1, ABa2, ABb1, ABb2, ABc1, ABc2, ABd1, and ABd2. All modules with analog measurement bus access can be switched to the analog measurement bus by means of coupling relays. These coupling relays are located directly on the analog measurement bus connector of the module so that the capacitive load of the analog measurement bus resulting from the module with the coupling relay open remains minimal.

After the coupling relay, the global analog measurement bus is continued on the module as a *local analog bus*. The measurement inputs or signal outputs of the relevant module can be connected with the local analog bus by means of a relay matrix.

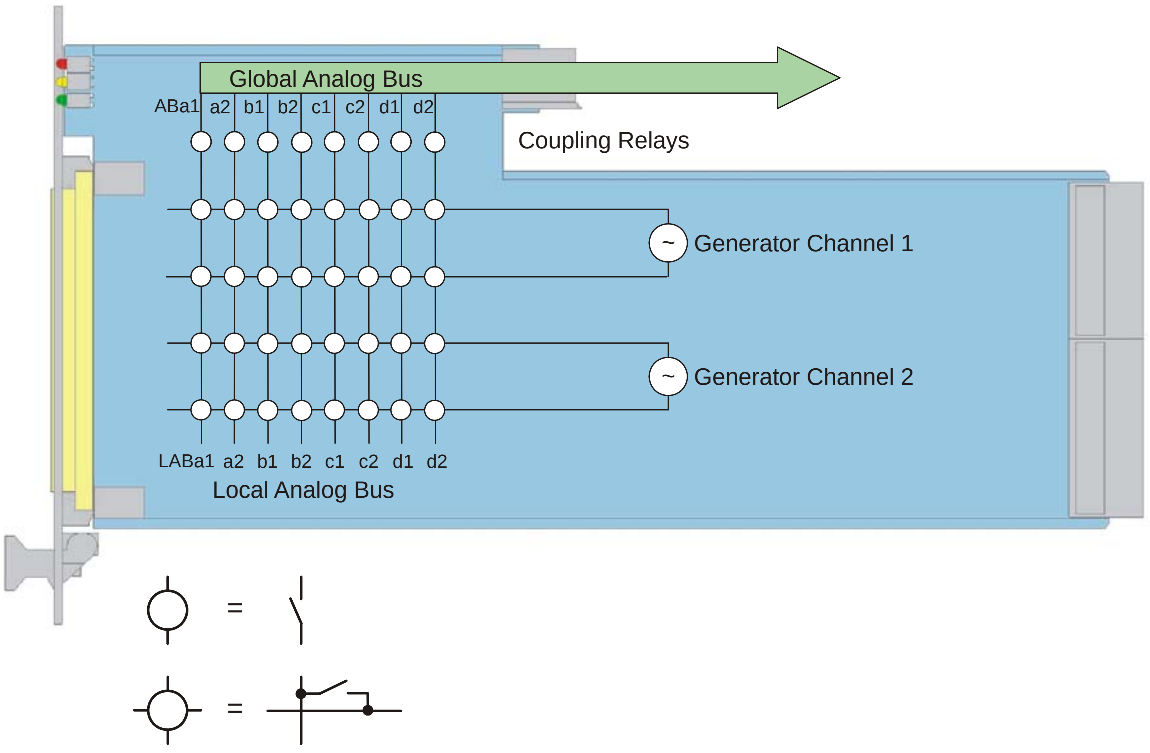


Figure 8-2 Analog measurement bus access via coupling relay

NOTE:

When connecting signals via the analog measurement bus or local analog bus, please note that they are approved only for voltages up to 125 VDC and currents up to 1 A.

Higher currents can be connected directly with the UUT through modules specially designed for that purpose (R&S TS-PSU, R&S TS-PSU12 R&S TS-PSM1, R&S TS-PSM2).

8.3 Configuration files

A test program contains switching commands that consist of a combination of *channel names* and *switching operations*.

The channel names in the test program correspond to names as they appear in the test specification or in the schematics of the UUT. These names are *logical channel names*. The software converts these names into *physical channel names*, i.e. into channel names as they are "understood" by stimulus, measurement, or switch modules.

Physical channel names are assigned to logical channel names in a UUT-specific *channel table*. This channel table is stored along with other information in a configuration file. It describes how connections from the UUT with the test system are made, i.e. it describes adapter wiring.

Using configuration files offers the advantage that the test program can concentrate on the actual measurement task. The adapter wiring and system configuration are defined outside of the test program in the configuration files.

NOTE:

Use of configuration files makes it possible to modify the adapter wiring or port the test program to a system with a different configuration without having to change the test program.

The R&S GTSL software uses two configuration files: one for the physical layer and the other for the application layer. The general layout of these two files is described in Chapter 5: Configuration Files.

8.3.1 Physical layer

For each test system there is exactly one file that describes the physical layer, i.e. the configuration of the system. This file's name is PHYSICAL.INI and it resides in directory ..\GTSL\Configuration. It contains the following information:

- Which hardware modules are present in the system?
- How are the hardware modules addressed?
- What software is responsible for the hardware modules?
- Options for device drivers, for example simulation mode
- Optionally a system-specific channel table

This file must be adjusted every time a change is made to the system configuration.

8.3.1.1 Example of a PHYSICAL.INI file

The following example shows a segment of a PHYSICAL.INI file. Some entries with no relevance to this chapter have been left out.

```
[device->PSAM]
Type         = PSAM
ResourceDesc = PXI1::10::0::INSTR
DriverDll    = rspsam.dll
DriverPrefix = rspsam
DriverOption = "Simulate=0,RangeCheck=1"

[device->PMB_10]
Type         = PMB
ResourceDesc = CAN0::0::1::10
DriverDll    = rspmb.dll
DriverPrefix = rspmb
DriverOption = "Simulate=0,RangeCheck=1"

[device->ABUS]
;analog measurement bus pseudo-device
;used by ROUTE, SWMGR and EGTSL
Type         = AB

[io_channel->system]
.DMM_HI      = PSAM!DMM_HI
.DMM_LO      = PSAM!DMM_LO
```

There is a [device->*Name*] section for each hardware module (device). There are no constraints on *Name*, but it must be unique within PHYSICAL.INI. The “Type” entry that defines the module type must be present for each device. The “ResourceDesc” entry must also be present. The software is able to access the module through this entry. The only exception is the pseudo device ABUS, which stands for analog measurement bus.

A system-specific channel table may optionally be present. On the left side it contains the logical channel name as it is permitted to occur in switching commands of the test programs. The name of the hardware module and the physical channel name in the form expected by the device driver of the corresponding module type appear on the right side (see section 8.3.4.)

8.3.2 Application layer

This configuration file is usually created individually for each UUT or test program. It can be assigned any name and be placed in any directory. For ease of comprehension, the file name APPLICATION.INI is used for this configuration file in the manual. It contains the following information:

- Which hardware modules are required for the test program?
- Options for R&S GTSL libraries, for example simulation, tracing
- The application-specific channel table

This information is combined in a *bench*. A *bench* thus defines which physical resources of the system are required for a UUT in what way.

An APPLICATION.INI file may also contain more than one bench and more than one channel table if multiple UUTs of the same type will be tested, for example in a panel test (see Section8.3.4).

8.3.2.1 Example of an APPLICATION.INI file

```
[bench->test]

; hardware modules
DigitalMultimeter = device->PSAM
FunctionGenerator = device->PFG
SwitchDevice1     = device->PSAM
SwitchDevice2     = device->PMB_10
SwitchDevice3     = device->PFG
AnalogBus         = device->ABUS
```

```
; options
Simulation        = 0
Trace             = 0

; link to channel table
AppChannelTable   = io_channel->test

; channel table
[io_channel->test]
INPUT             = PMB_10!P1
GND               = PMB_10!P2
OUTPUT            = PMB_10!P3
MONITOR           = PMB_10!P65
```

In the first section, the hardware modules required for the test are listed. The various R&S GTSL libraries recognise the devices they should work with by means of the keywords on the left side. The right side contains references to the corresponding device entries in PHYSICAL.INI.

The second section includes options for the R&S GTSL libraries.

The third section contains a reference to the channel table that will be used.

The channel table in the fourth section contains (on the left side) logical channel names as they occur in the switching commands of the test program. The name of the device and the physical channel name in the form expected by the device driver of the corresponding module type appear on the right side.

8.3.3 Special entries for switched connections

There are three libraries in R&S GTSL that can make switched connections:

- The Signal Routing Library ROUTE.DLL (see also Section 7.1.11)
- The Switch Manager SWMGR.DLL (see also Section 7.1.12)
- The library for In-Circuit Test ICT.DLL (see also Section 7.2.1)

All three libraries use the same entries of APPLICATION.INI in terms of switched connections. The following table shows an overview of keywords.

Keyword	Value	Description
SwitchDevice<i>	String	*Mandatory entry* Refers to a device entry section of a switch device in PHYSICAL.INI. <i> stands for a number from 1,2,3,…,n. The numbers must be assigned in ascending order without gaps. <i> may be omitted in case it is 1.
AnalogBus	String	*Mandatory entry* Refers to the device section of the analog bus in PHYSICAL.INI.
AppChannelTable	String	*Mandatory entry* Refers to a section [io_channel->…] with defined channel names in APPLICATION.INI.
SwitchSettings	String	*Optional entry* Refers to a section [switch->…] with defined switch settings in APPLICATION.INI
Simulation	0 / 1	*Optional entry* Blocks the simulation of the entered devices (value = 0). Enables simulation of the entered devices (value = 1). Default = 0
Trace	0 / 1	*Optional entry* Blocks the tracing function of the library (value = 0). Enables the tracing function of the library (value = 1). Default = 0
ChannelTableCaseSensitive	0 / 1	*Optional entry* The channel names in the channel table are treated case-sensitive (value = 1) or case-insensitive (value = 0). Default = 0
SignalRoutingDisplay	0 / 1	*Optional entry* Displays a window with actual signal routing information (value=1). Default = 0

8.3.3.1 SwitchDevice<i>

These mandatory entries are references to the corresponding device sections in PHYSICAL.INI. Based on the mandatory “Type” entry in PHYSICAL.INI, the libraries determine whether the corresponding module type is supported. The entries “ResourceDesc”, “DriverDLL” and “DriverPrefix” must also be present.

The suffix <i> represents a sequential numbering, i.e. SwitchDevice1, SwitchDevice2, etc. Instead of “SwitchDevice1”, “SwitchDevice” can also be written.

ICT Library / R&S EGTSL: Only SwitchDevice entries of type PMB are considered. All others are ignored.

Switch Manager: The Switch Manager supports the same module types as the Signal Routing Library plus modules R&S TS-PMA and R&S TS-PRL1 of the R&S ClassicTSVP.

Signal Routing Library: The Signal Routing Library supports the following module types:

Type	Module designation
PAM	R&S TS-PAM Analyzer Module
PDFT	R&S TS-PDFT Digital Functional Test Module
PFG	R&S TS-PFG Function Generator Module
PIO2	R&S TS-PIO2 Analog/Digital IO Module 2
PMB	R&S TS-PMB Matrix Module
PSAM	R&S TS-PSAM Analog Source and Measurement Module
PSM1	R&S TS-PSM1 Power Switching Module 1
PSM2	R&S TS-PSM2 Multiplex/Switch Module 2
PSU	R&S TS-PSU Power Supply/Load Module
PSU12	R&S TS-PSU12 Power Supply/Load Module 12V
PSYS1	R&S TS-PSYS1 System Module
PSYS2	R&S TS-PSYS2 System Module
IVI_SWITCH	Any generic switching module that provides an IVI-C driver of the IviSwtch class

Table 8-2 Module types supported by the Signal Routing Library

8.3.3.2 AnalogBus

This mandatory entry is a reference to the pseudo device “ABUS” of PHYSICAL.INI.

Switch Manager: The AnalogBus entry is optional. If it is not present, no connections via the analog measurement bus are possible.

8.3.3.3 AppChannelTable

This mandatory entry is a reference to the application-specific channel table. See also Section 8.3.4.

Switch Manager: The AppChannelTable entry is optional. If it is not present, switched connections can only be made with physical channel names.

8.3.3.4 SwitchSettings

This optional entry is a reference to the switch settings. Switch settings are pre-defined switching commands. They are supported only by the Signal Routing Library. Refer to chapter 8.4.3.

8.3.3.5 Simulation

This optional entry turns simulation mode of the libraries on and off. There is no access to hardware in simulation mode and the device drivers are not loaded. Since the Signal Routing Library must have access to the device drivers to search for paths, however, it behaves differently in simulation mode. This means it is unable to report errors if connections are not possible.

8.3.3.6 Trace

This optional entry turns tracing of the libraries on and off. When tracing is activated, the libraries write information to a file or screen window during execution. The Resource Manager library makes various options available for tracing; see Section 7.1.8.

8.3.3.7 ChannelTableCaseSensitive

This optional entry determines whether upper/lower case should be distinguished for logical channel names. If the entry is missing or has a value of 0, there is no distinction between upper and lower case. The channel names “Input” and “INPUT” will be treated identically.

If the relevant entry has a value of 1, “Input” and “INPUT” represent two different channels.

8.3.3.8 SignalRoutingDisplay

This optional entry indicates whether the Signal Routing Library displays a window in which the current switched connections are represented. See also Section 8.4.5.

8.3.4 Channel tables

R&S GTSL libraries for switched connections use two channel tables to assign the logical channel names used in the test program to physical channel names of the test system:

- the application-specific channel table that is referenced in APPLICATION.INI with the keyword “AppChannelTable”.
- optionally a system-specific channel table stored in PHYSICAL.INI in the [io_channel->system] section.

The Signal Routing Library and Switch Manager Library combine both tables into a general channel table. The ICT library / R&S EGTSL loads only the application-specific channel table.

The two tables are identical in general structure. They have an entry for each logical channel name to assign a physical channel name to it, for example:

```
GND = PMB_10!P2
```

The logical channel name on the left side must be unique. This means that it must only occur once in the two channel tables. The “ChannelTableCaseSensitive” option determines whether or not to distinguish between upper and lower case.

Logical channel names must be no more than 80 characters long and may only contain the following characters:

"A"..."Z"	uppercase letter
"a"..."z"	lowercase letter
"0"..."9"	digit
"_"	underscore
"."	decimal point/period
"!"	exclamation mark
"#"	number sign
"$"	dollar sign
"%"	percent
"&"	ampersand
"*"	asterisk
"+"	plus
"-"	minus
"/"	slash
"\"	backslash
":"	colon
"?"	question mark
"@"	at sign
"^"	caret
"|"	vertical bar
"~"	tilde
"("	opening paranthesis
")"	closing paranthesis
"{"	opening curly brace
"}"	closing curly brace
"["	opening square bracket
"]"	closing square bracket
"<"	opening angle bracket/less than
">"	closing angle bracket/greater than

Table 8-3 Character set for logical channel names

The physical channel name (made up of the device name and the device-specific channel name) is on the right side. Only device names that are referenced in a “SwitchDevice<i>” entry can be used. The “device->” prefix of this entry has been omitted to make the channel table easier to read.

The device name is separated from the device-specific channel name by an exclamation mark. This is a channel name that is accepted by the device driver. For the specific name, see the device driver documentation (usually the description of function *xyz_Connect*).

Although logical channel names must be unique, the same does not apply to physical channel names. It is permissible to assign several logical channel names to the same physical channel (*alias names*).

The following rules apply to physical names of analog measurement bus lines:

- The physical names of global analog measurement bus lines are ABUS!ABa1 to ABUS!ABd2. ABUS is the pseudo device analog measurement bus of type = AB.
- The physical names of the local analog measurement bus lines are *device*!LABa1 to *device*!LABd2. In this case *device* stands for a device of any other type. This rule also applies if the device driver does not accept the physical names “LABxy”, but only accepts “ABxy” (for example R&S TS-PSAM).

Example:

```
ABa1 = ABUS!ABa1
PSAM.LABa1 = PSAM!LABa1
PSM1.LABA1 = PSM1!LABa1
```

Channel attributes can optionally be assigned for channels. Channel attributes separated by commas are appended to the physical channel names, for example:

```
.ABa1 = ABUS!ABa1,nonrouting
```

Channel attributes are described in Chapter 8.4.4.

A *comment* can optionally be provided for a channel. It is introduced by a semicolon. Anything after the semicolon to the end of the line is comment:

```
VCC = PMB_10!P75 ; +5 V supply
```

8.3.4.1 System-specific channel table

The system-specific channel table is optional. It is stored in PHYSICAL.INI in the [io_channel->system] section.

The channel names that are required in many test programs and are not application-specific are defined in the system-specific channel table. Examples include

- Inputs of measuring devices, for example the DMM_HI and DMM_LO inputs of the R&S TS-PSAM multimeter.
- Outputs of stimulus devices, for example the CH1_HI and CH1_LO outputs of the R&S TS-PSAM function generator.
- Channels of switch modules that have a fixed connection with external devices, for example power channels of the R&S TS-PSM1 module that are connected with external power supplies.

The system-specific channel table thus describes the fixed wiring of the system.

To ensure that logical channel names are unique and to avoid duplicate names with the application-specific channel table, it is recommended that all logical channel names of the system-specific channel table begin with a period.

8.3.4.2 Application-specific channel table

The application-specific channel table is stored in APPLICATION.INI in the [io_channel->*Name*] section. There are no constraints on *Name*, but it must be unique within APPLICATION.INI.

Channel names that are required especially for the UUT are defined in the application-specific channel table. These are:

- Measurement points on the UUT that are connected with switch modules.
- Measurement points on the UUT that are connected with local multiplexers of measurement or stimulus modules.

The application-specific channel table therefore describes the adapter wiring for the relevant UUT.

Entries of the application-specific channel table may also contain logical names of the system-specific channel table on the right side.

8.4 Signal Routing Library

8.4.1 Example of a switched connection

The following example demonstrates the functionality of the Signal Routing Library by way of a simple switched connection task.

A signal is applied to the input of an amplifier and then measured at the output of the amplifier.

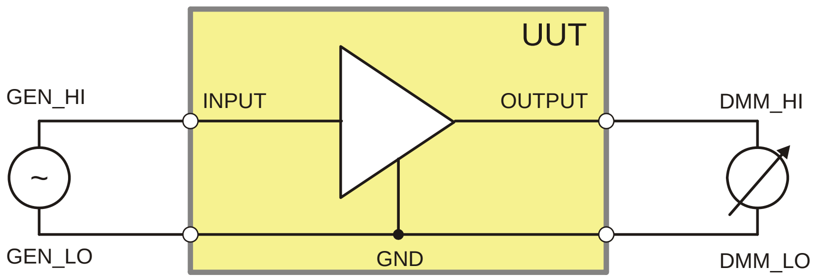


Figure 8-3 Measurement task

The PHYSICAL.INI file contains entries for devices PSAM, PFG and PMB_10. No system channel table will be used in this example. The corresponding APPLICATION.INI appears as follows:

```
[bench->test]
DigitalMultimeter = device->PSAM
FunctionGenerator = device->PFG
SwitchDevice1     = device->PSAM
SwitchDevice2     = device->PMB_10
SwitchDevice3     = device->PFG
AnalogBus         = device->ABUS
AppChannelTable   = io_channel->test

[io_channel->test]
; UUT channels
INPUT             = PMB_10!P1
GND               = PMB_10!P2
OUTPUT            = PMB_10!P3
MONITOR          = PMB_10!P65  ; used in later  example
; system channels
GEN_HI            = PFG!CH1_HI
GEN_LO            = PFG!CH1_LO
DMM_HI            = PSAM!DMM_HI
DMM_LO            = PSAM!DMM_LO
```

Devices PSAM, PFG, and PMB_10 are required for switching. Therefore they are entered as SwitchDevice<i>. The channel table contains the channel names of the UUT and the channels within the system that will be connected with the UUT.

Devices PSAM and PFG are also used as a digital multimeter and function generator (libraries DMM.DLL and FUNCGEN.DLL). They are therefore also entered as DigitalMultimeter and FunctionGenerator respectively.

The following switching commands are executed to set up the switched connection:

```
GEN_LO > GND
GEN_HI > INPUT
GND > DMM_LO
OUTPUT > DMM_HI
```

For each switching command, the Signal Routing Library searches for a suitable free analog measurement bus and sets up the following switched connection:

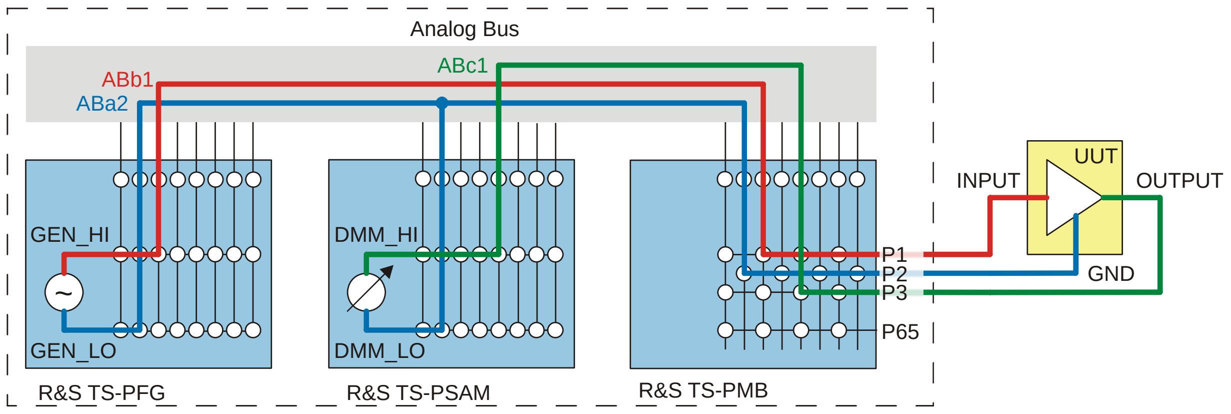


Figure 8-4 Switched connection for measurement task

The corresponding test program is roughly as follows. The sections of code for handling errors have been omitted for the sake of clarity:

```
// Variables
short errorOccurred;
long  errorCode
char  errorMessage[GTSL_ERROR_BUFFER_SIZE];
long  resourceId;

// setup libraries
RESMGR_Setup ( 0, "physical.ini", "testApplication.ini",
               &errorOccurred, &errorCode, errorMessage  );

ROUTE_Setup ( 0, "bench->test", &resourceId,
              &errorOccurred, &errorCode, errorMessage  );
```

```
// connect generator and DMM
ROUTE_Execute ( 0, resourceId,
                "GEN_LO > GND, GEN_HI > INPUT,  GND > DMM_LO, OUTPUT >DMM_HI",
                &errorOccurred, &errorCode, errorMessage  );

// apply generator signal and measure output (not  shown here)
// ...

// disconnect all
ROUTE_Execute ( 0, resourceId, "||", &errorOccurred, &errorCode,errorMessage );

// Close libraries
ROUTE_Cleanup ( 0, resourceId, &errorOccurred, &errorCode, errorMessage );
RESMGR_Cleanup ( 0, &errorOccurred, &errorCode,  errorMessage );
```

At the beginning of the program the required R&S GTSL libraries are initialised. The Resource Manager must always be called first. `RESMGR_Setup` loads the two configuration files “physical.ini” and “testApplication.ini”. `ROUTE_Setup` then loads the channel tables and prepares the hardware modules for use.

Then come the actual switching commands with a `ROUTE_Execute` call and the remainder of the test program (not shown here). This part of the program can be repeated several times if several UUTs need to be tested.

At the end of the test program, the corresponding cleanup function must be called for each setup function that was called at the beginning. `RESMGR_Cleanup` is the last R&S GTSL function that is called.

8.4.2 Switching commands

The ROUTE_Execute function of the Signal Routing Library performs switching commands. The following switching commands are possible:

Switching command	Function
a > b	Connect channels a and b
a | b	Disconnect channels a and b
a || b	Disconnect all connections in the path between channels a and b
a ||	Disconnect all connections previously made with channel a.
||	Disconnect all existing connections
%	Disconnect all obsolete connections
#s	Make switched connection of switch setting s
#s |	Break switched connection of switch setting s

Table 8-4 Simple switching commands

Switching command	Function
#s ||	Break switched connection of switch setting s; all connections along the path are disconnected.
?n	Wait n milliseconds
?#	Wait for debounce of all switch modules
?#n	Wait for debounce of all switch modules with timeout n milliseconds
,	Delimiting character for switching commands
;	Comment at the end of a switching command

Table 8-4 Simple switching commands

“a” and “b” stand for logical channel name which must be present in the application-specific table or system channel table or for system names. “s” stands for the name of a switch setting (see Section 8.3.3.4) and “n” stands for a real numeric literal.

Multiple switching commands separated by commas can be combined to form a single switching command. A comment may optionally be placed at the end of a switching command. Comments are introduced by a semicolon.

The following compound switching commands are available to simplify entry of complex switching commands. They are separated into simple commands during processing as shown in the table below:

Switching command	Corresponds to simple commands	Function
a > b > c > d	a > b, b > c, c > d	Extended connection
a | b | c | d a || b || c || d	a | b, b | c, c | d a || b, b || c, c || d	Multiple disconnection
a * b * c * d	a > b, a > c, a > d	Star connection

Table 8-5 Compound switching commands

Compound switching commands contain the same simple switching command multiple times. It is not permitted to combine switching commands of different types to form a compound switching command.

8.4.2.1 Channel names in switching commands

Channel names used in switching commands may be:

- Logical channel names from the application-specific channel table
- Logical channel names from the system channel table
- System names

The logical channel names are defined in the corresponding channel tables (see Section 8.3.4).

System names are channel names that have been identified to the Signal Routing Library without the names having to be explicitly defined in the channel tables. System names always start with a “$” sign.

System names	Description
$ABa1, $ABa2 $ABb1, $ABb2 $ABc1, $ABc2 $ABd1, $ABd2	System names for analog measurement bus lines of the global analog measurement bus.
$LABa1, $LABa2 $LABb1, $LABb2 $LABc1, $LABc2 $LABd1, $LABd2	System names for the local analog measurement bus lines of a device.

Table 8-6 System names

The system names for the global analog measurement bus stand for physical channel names ABUS!ABa1, ABUS!ABa2, etc and are unique within the entire system. By contrast, system names of local analog measurement buses may not be assigned uniquely to a device if multiple devices are present in the system. The assignment is based first on the context of the switched connection command. Example:

```
DMM_HI > $LABa1 > $ABa1
```

Since DMM_HI is assigned to the PSAM device, i.e. the physical channel name is PSAM!DMM_HI, $LABa1 is also assigned to the PSAM device.

```
$LABa1 > $LABa2
```

In this case no assignment can be made to a device. An error is reported.

```
DMM_HI > $LABa1 > INPUT
```

In this case the context is different on the left and right side. $LABa1 can be assigned to either the PSAM device or the PMB_10 device. The software is not capable of deciding which local local analog measurement bus is meant. An error is also reported in this case.

If a logical channel name contains any special characters reserved for switching commands, it must be enclosed in single quotes. This rule applies for the following character set:

> | % * # ? $ -

8.4.2.2 Connecting channels

Command “**a > b**” connects channel a with channel b.

The connection can be routed either directly or via local and global analog measurement bus lines.

There is a *direct connection* present if channel a and channel b can be connected by closing a single relay. Direct connections must have either both channels on the same device or else a global analog measurement bus as one of the two channels.

The Signal Routing Library is able to set up complex switched connections via automatic routing. Local and global analog measurement bus lines are used for automatic routing to connect the two channels together.

An automatically routed connection may always be seen as a sequence of direct switched connections:

```
OUTPUT > DMM_HI
OUTPUT > $LABb1 > $ABb1 > $LABb1 > DMM_HI
```

8.4.2.3 Disconnecting channels

Commands “**a | b**” and “**a || b**” disconnect the existing connection between channel a and channel b.

The difference between the two commands is that “a | b” disconnects the connection at precisely one point and allows partial connections to remain in place. Command “a || b” opens all connections along the switching path. Partial connections that remain intact after channels are disconnected are called *obsolete connections*.

Command “a | b” is more efficient, since only one connection needs to be opened. It may be possible to use the partial connections that remain intact in a subsequent switching command. That would minimise the number of relays to be switched in this case as well. This procedure avoids unnecessary switching cycles of relays, thus extending their service life.

It is only possible to disconnect channels that were previously connected with each other in the same manner, i.e. the command “a | b” must be preceded by the command “a > b”, otherwise a warning will be reported. It is also permissible to exchange the order of channels. In other words, the command “b | a” is also permitted.

NOTE:

After disconnecting connections, to ensure that all connections are actually disconnected before setting up new connections, it is recommended to use the command “?#” (Wait for Debounce).

8.4.2.4 Other disconnect commands

The command “**a ||**” disconnects all existing connections that had previously been made with channel a. At the same time, all connections along the switching paths are opened. This isolates channel a from all other channels.

Example:

```
a > b > c, d > a > e, a||
```

Channel a was connected with b, d and e. Disconnect command a|| thus corresponds to commands

```
a || b, a || d, a || e
```

The command “**||**” disconnects all existing active and obsolete connections. It resets the relays of all devices administered by the Signal Routing Library. This command resets all relays on the module responsible for the switched connection, including the coupling relays. Relays that create the ground reference of stimulus and measurement devices are not affected. These are not considered part of the switched connection and therefore cannot be either switched or opened by the Signal Routing Library.

The command “%” breaks all obsolete connections. For more information, see Section 8.4.6.7.

NOTE:

After disconnecting connections, to ensure that all connections are actually disconnected before setting up new connections, it is recommended to use the command “?#” (Wait for Debounce).

The two commands “||” and “%” affect all switch modules that are configured in any bench as SwitchDevice<i>.

8.4.2.5 Switch setting commands

Switch settings are switching commands that are stored in APPLICATION.INI under a user-defined name and can be called and executed in the test program with their names. Switch settings are explained in detail in Section 8.4.3.

8.4.2.6 Wait commands

The purpose of wait commands is to delay execution of the switched connection for a certain amount of time. This may be necessary to ensure that a connection has actually been made before performing a measurement, or to ensure that one connection has been opened before another one is closed (“Break-Before-Make”).

The Signal Routing Library recognises two types of wait commands:

- Fixed wait time
- Wait until all relays have switched and are debounced

The command “?n” delays all subsequent switching commands by n milliseconds. The command “?#” waits until all previously switched relays are debounced. Optionally in the command “?#n”, a maximum time n may be specified in milliseconds. If all relays are not debounced after this time, the command is aborted with an error. If no maximum time is specified, the default value of 100 ms is used.

In both cases, “n” stands for a real numeric that must be greater than 0 with a maximum value of 10000 (10 seconds).

```
POWER | P1, ?#, POWER > P2, ?#
```

This switching command disconnects POWER from P1 and waits until the connection is debounced, i.e. confirmed opened, before making the connection to P2. Then the function waits until the new connection has been set up before the ROUTE_Execute function returns and e.g. a voltage measurement can be performed.

8.4.2.7 Compound commands

Individual switching commands can be combined to form longer switching commands. The commands are separated from each other by commas:

```
GEN_LO > GND, GEN_HI > INPUT,  DMM_LO > GND, ?#
```

8.4.2.8 Comment

A comment in a switching command is introduced by a semicolon. All following characters are ignored by the command interpreter. Comments are especially helpful in switching commands that are saved as switch settings in APPLICATION.INI.

8.4.3 Switch settings

Switch settings are switching commands that are stored in APPLICATION.INI under a user-defined name and can be called and executed in the test program with their names.

The advantage of switch settings is being able to save very complex switched connections under a meaningful name. These switched connections can be set up and disconnected again in the test program by passing their name to the switching command.

Switch settings are checked for correct syntax when the Signal Routing Library is loaded and prepared for runtime optimisation. This makes it possible for them to run faster in `ROUTE_Execute` than the corresponding directly transferred switching command.

8.4.3.1 Entries in APPLICATION.INI

Like channel tables, switch settings are saved in APPLICATION.INI in the [switch->*Name*] section. There are no constraints on the *Name* entry, but it must be unique within the APPLICATION.INI. The name of this section is referenced in the bench with the keyword SwitchSettings:

```
[bench->test]
DigitalMultimeter = device->PSAM
FunctionGenerator = device->PFG
SwitchDevice1     = device->PSAM
SwitchDevice2     = device->PMB_10
SwitchDevice3     = device->PFG
AnalogBus         = device->ABUS
AppChannelTable   = io_channel->test

SwitchSettings    = switch->test
```

On the left side, the switch setting table contains the names of the switch settings, which must begin with a “#” sign. The switching command is on the right side.

The switch setting name on the left side must be unique. It must be no more than 80 characters long and may only contain the following characters:

“#”	Switch setting prefix
“A” ... “Z”	Upper case characters
“a” ... “z”	Lower case characters
“0” ... “9”	Numbers
“_”	Underscore
“.”	Decimal point

Table 8-7 Character set for switch setting names

The character “#” introduces the name and is only permitted as the first character.

The switching command is on the right side. It may contain simple and compound switching commands as well as a comment. However, it may not contain additional switch setting commands.

The maximum length of the switching command is 260 characters. Switching commands that contain more than 260 characters must be broken up into several lines. This can be done by entering a switch setting with the same name in the following line and continuing the switching command there.

Example:

```
[switch->test]

#ConnectGenerator = GEN_LO > GND,  GEN_HI > INPUT  ; Generator  HI and LO to UUT

#ConnectDMM       = DMM_LO > GND,  DMM_HI > OUTPUT ; DMM HI and LO to UUT

#ConnectAll       = GEN_LO > GND,  GEN_HI > INPUT,
#ConnectAll       = DMM_LO > GND,  DMM_HI > OUTPUT,
#ConnectAll       = ?# ; Connect generator and DMM to UUT and wait for debounce
```

8.4.3.2 Making switch settings

Switch settings are performed by specifying the name in the switching command:

```
#ConnectAll
```

Switch settings can be combined with switching commands and additional switch settings:

```
#ConnectGenerator, #ConnectDMM, DMM_LO | GND,  ?#
```

Appending one of switching commands “|” or “||” breaks the switched connection of switch settings. With these commands, all “>” commands for connecting channels are replaced by “|” or “||”:

```
#ConnectAll |
```

If the switch setting already contains commands for disconnecting the connections, they remain unchanged, but the `ROUTE_Execute` function returns a warning.

8.4.4 Channel attributes

Channel attributes define certain properties of a channel. They are optional and can be entered in the channel tables. Channel attributes separated by commas are appended to the physical channel names, for example:

```
.ABa1 = ABUS!ABa1,nonrouting
```

8.4.4.1 Channel attribute “nonrouting”

The channel attribute *nonrouting* can be used for global and local analog measurement bus lines. If it is specified, it makes it impossible for the analog measurement bus line of the Signal Routing Library to be used for automatic path search.

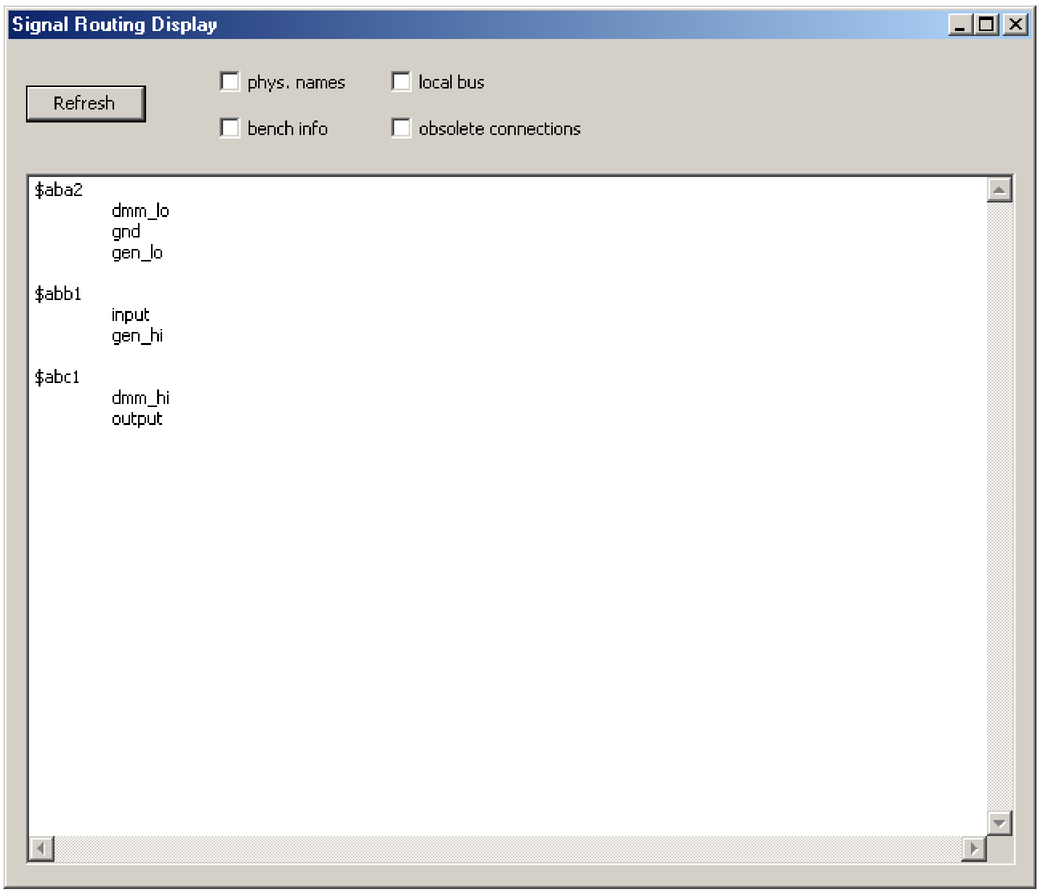
This attribute can be used, for example, if an analog measurement bus line has a fixed connection with a signal fed in externally, which therefore must not be connected automatically with other signals.

8.4.5 Display switched connection

During the development and test phase of a test program, it is helpful to be able to display the current state of switched connections. This display can be activated by specifying the option

```
SignalRoutingDisplay = 1
```

in APPLICATION.INI. If the display is activated, a window appears when `ROUTE_Setup` is called. The window continues to be displayed until `ROUTE_Cleanup` is called. The window can be minimised during this time, but cannot be closed.



```
$aba2
          dmm_lo
          gnd
          gen_lo

$abb1
          input
          gen_hi

$abc1
          dmm_hi
          output
```

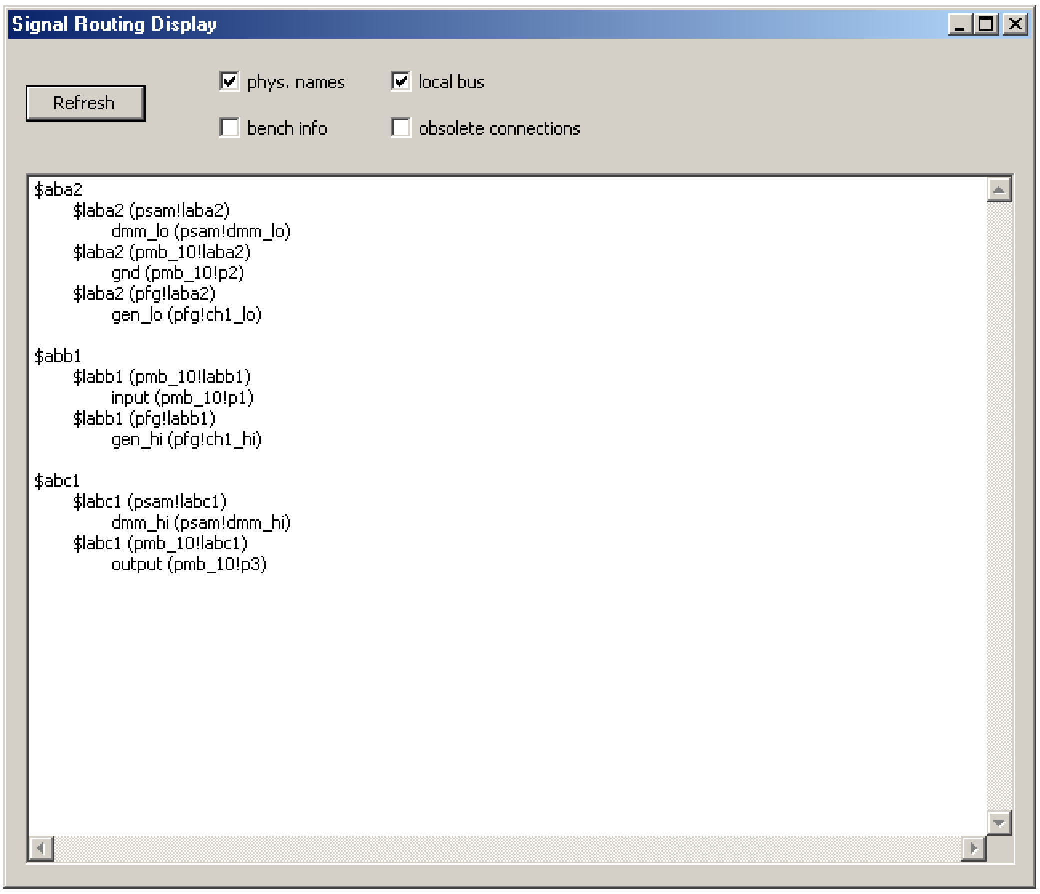
Figure 8-5 Signal Routing Display

Clicking on the “Refresh” button or pressing the F5 key updates and shows current switched connections.

The scope of information that appears can be changed by selecting the options in the upper display area:

phys. names	displays in addition the physical channel names
local bus	displays in addition the connection via local analog measurement buses
bench info	displays in addition the name of the bench
obsolete connections	displays in addition existing obsolete connections

The switched connection is displayed sorted according to analog measurement bus lines. For each analog measurement bus, a list of channels connected to it is shown. The example above shows the switched connection of Figure 8-4.



[✓] phys. names [✓] local bus
[ ] bench info [ ] obsolete connections

```
$aba2
        $laba2 (psam!laba2)
                dmm_lo (psam!dmm_lo)
        $laba2 (pmb_10!laba2)
                gnd (pmb_10!p2)
        $laba2 (pfg!laba2)
                gen_lo (pfg!ch1_lo)

$abb1
        $labb1 (pmb_10!labb1)
                input (pmb_10!p1)
        $labb1 (pfg!labb1)
                gen_hi (pfg!ch1_hi)

$abc1
        $labc1 (psam!labc1)
                dmm_hi (psam!dmm_hi)
        $labc1 (pmb_10!labc1)
                output (pmb_10!p3)
```

Figure 8-6 Display with local analog measurement buses and physical channel names

The indentation of channel names indicates over how many sub-paths the channels are connected with each other.

8.4.6 Switched connection algorithms

This chapter treats switched connection algorithms, i.e. the rules that define how switched connections in the Signal Routing Library are set up and broken.

8.4.6.1 Connecting channels

There are three possible results when two channels are to be connected:

- A direct connection between the channels is possible
- a connection via local/global analog measurement is possible
- No connection is possible

To make a *direct connection* only a single switching process is required. In technical terms, a direct connection can be made by calling the `xyz_Connect` function of the device driver. If a direct connection between two channels is possible, the Signal Routing Library will create it.

In direct connections, the two channels are always on the same module. For modules with analog measurement bus access, one of the two channels may also be on the analog measurement bus.

On third-party modules, i.e. modules of type IVI_SWITCH, the Signal Routing Library can only apply direct connections.

If a direct connection is not possible, the Signal Routing Library can search for a connection through local and global analog measurement buses.

If no such connection is found either, the switched connection cannot be made and the `ROUTE_Execute` function returns with an error.

If two channels need to be connected to each other and there is already a connection between them, the `ROUTE_Execute` returns with a warning. The current connection is not changed.

8.4.6.2 Routing via analog measurement buses

The Signal Routing Library can make connections automatically via free local and global analog measurement bus lines. This makes it easy to connect two channels on different modules together. The coupling relays are switched automatically.

The Signal Routing Library first searches for analog measurement buses with which a connection is potentially possible. Not any channel may be connected with any analog measurement bus. The following table gives an overview of the number of analog measurement bus access points per channel:

Module type	Channels	Analog measurement bus access points per channel
	CHA1_HI, …, CHA4_HI, CHB1_HI, …, CHB4_HI	4
R&S TS-PFG	CH1_HI, CH1_LO, CH2_HI, CH2_LO	8
R&S TS-PIO2	CH1_IN, … CH16_IN	2
R&S TS-PMB	P1, …, P90 IL1, …, IL3	4 8
R&S TS-PSAM	DMM_HI, DMM_LO, DMM_SHI, DMM_SLO	8
R&S TS-PSM1	CH1com, …, CH16com, CH1no, …, CH16no LPBA, …, LPBD, IL1com, IL2com, IL1no, IL2no	1 or 2
R&S TS-PSM2	CH1_HI, …, CH8_HI, CH1_LO, …, CH8_LO	2
R&S TS-PSU R&S TS-PSU12	CH1_HI, CH1_LO, CH1_SHI, CH1_SLO CH2_HI, CH2_LO, CH2_SHI, CH2_SLO	4 4

Table 8-8 Analog measurement bus access of various module types

It is possible there is already a connection with the desired signal to an analog measurement bus. In this case an attempt is made to make a connection to this analog measurement bus. Otherwise a free analog measurement bus is selected from potentially available buses and the connection is made through that bus.

NOTE:

Modules with few analog measurement bus access points should always be connected before modules with many analog measurement bus access points. This will ensure the routing algorithm is still able to find enough potential analog measurement buses for the switched connection.

The analog measurement bus line is selected to minimise “cost” as much as possible. The system-wide analog measurement bus is the most expensive resource, followed by the local analog measurement buses. There are special switching options for R&S TS-PMB modules that are explained in greater detail in Section 8.4.6.9.

8.4.6.3 Manual and automatic routing

Complex (i.e. non-direct) switched connections can be routed manually or automatically. *Manual routing* means the switching command contains exclusively direct connections:

```
GEN_HI > $LABa1 > $ABa1 > $LABa1 > INPUT
```

The connection `GEN_HI > $LABa1` is a direction connection, as is `$LABa1 > $ABa1`, etc.

It is also to perform the switching command with automatic routing. In this case the Signal Routing Library selects a suitable analog measurement bus line for the connection:

```
GEN_HI > INPUT
```

Which analog measurement bus line is suitable depends on the actual switching state of the system. Therefore it cannot be assumed that GEN_HI > INPUT will always select the same analog measurement bus line for automatic routing. If another signal is already assigned to $ABa1, for example, a different free analog measurement bus line must be found. Automatic routing is only possible via local and global analog measurement bus lines. Switched connections over local multiplexer or power buses can only be routed manually.

8.4.6.4 Manually and automatically routed channels

A channel that is explicitly listed in a switching command is said to be *manually routed*. By contrast, channels, which were automatically selected by the routing algorithm rather than being explicitly listed, are said to be *automatically routed*.

Connections to automatically routed channels are not permitted. Example:

```
GEN_HI > INPUT
POWER > $ABa1 > VCC
```

The first switching command establishes a connection between GEN_HI and INPUT. The second switching command connects the POWER signal via the analog measurement bus $ABa1 with the VCC channel of the UUT. Assuming the first switched connection automatically selects analog measurement bus $ABa1, the second command would result in a short circuit between GEN_HI and POWER. This connection is therefore not permissible and the `ROUTE_Execute` function reports an error.

On the other hand, if the first switching command is routed via $ABb1 and $ABa1 is still free, for example, the second switching command can be performed.

8.4.6.5 Multiple assignment of switching paths

In the following example, the generator signal from GEN_HI will be directed to the input of the UUT and also to a monitor output to connect an oscilloscope, for example. The corresponding switching commands are:

```
GEN_HI > INPUT
GEN_HI > MONITOR
```

The Signal Routing Library first sets up the connection GEN_HI > INPUT via a free analog measurement bus line, for example $ABa1.

The already existing switched sub-connection from GEN_HI to LABa1 of the R&S TS-PMB module can be used for the second connection GEN_HI > MONITOR. Then the Signal Routing Library only needs to make the connection between LABa1 and MONITOR.

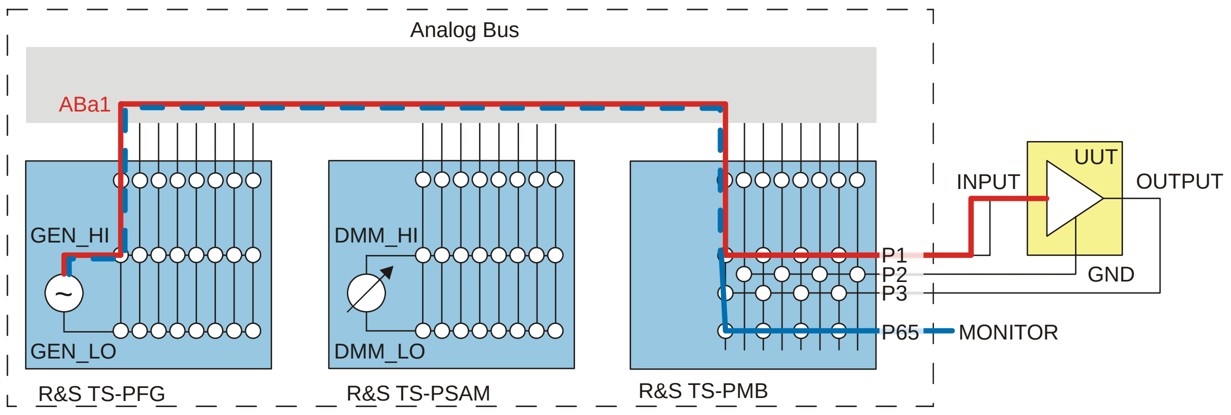


Figure 8-7 Multiple assignment of a switching path

The section between GEN_HI and LABa1 is now used by two switched connection paths. The Signal Routing Library administers a *reference counter* for each partial section of a switched connection. The counter shows how many switched connections use the partial section. The reference counters for the partial sections used in common (represented in the figure by red and blue dashed lines) are set by the switched connection GEN_HI > MONITOR to 2.

The reference counters are important when breaking the connection:

```
GEN_HI || INPUT
```

If GEN_HI is disconnected from INPUT, it must still be ensured that the connection GEN_HI to MONITOR remains intact. Therefore the sections that are used in common must not be disconnected.

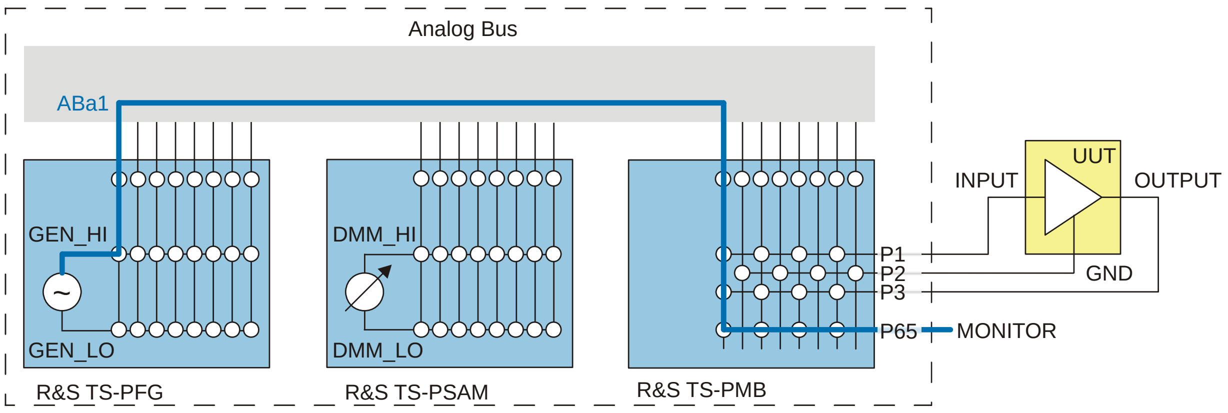


Figure 8-8 Multiple assignment after breaking the connection to INPUT

Disconnection causes all reference counters along the switched connection to be decremented, but only those whose reference counters are now 0 may be opened. The relays along the entire path of the switched connection are not opened until the other connection has also been disconnected:

```
GEN_HI || MONITOR
```

If a switching command is performed for an already existing connection, a warning is reported and the reference counter is not incremented:

```
GEN_HI > INPUT
GEN_HI > INPUT
```

The following command disconnects the connection because the reference counter has not been changed for the second switching command:

```
GEN_HI || INPUT
```

8.4.6.6 Disconnecting connections

Only connections that were previously set up with the same two channels can be disconnected with the commands “|” and “||”. Otherwise a warning is reported.

When connections are disconnected, multiple assignment of switching paths is taken into consideration. The connection can only be disconnected at this point (i.e. this relay can only be opened) if the reference counter of a section (i.e. of a relay) becomes zero.

The command “a || b” opens all relays along the path between a and b, provided their reference counters are not greater than 1.

The command “a | b” opens the connection at one point only. This command can be performed more quickly than “a || b” because only one switching process is necessary. Connections that are currently no longer required remain intact. These connections are referred to as *obsolete connections*, in contrast to *active connections*.

8.4.6.7 Obsolete connections

An obsolete connection remains intact if an automatically routed connection is opened with the switching command “a | b”. Example:

```
GEN_HI > INPUT
GEN_HI | INPUT
```

Assuming the first switching command sets up a connection via $ABa1, there are several ways for the second switching command to disconnect the connection:

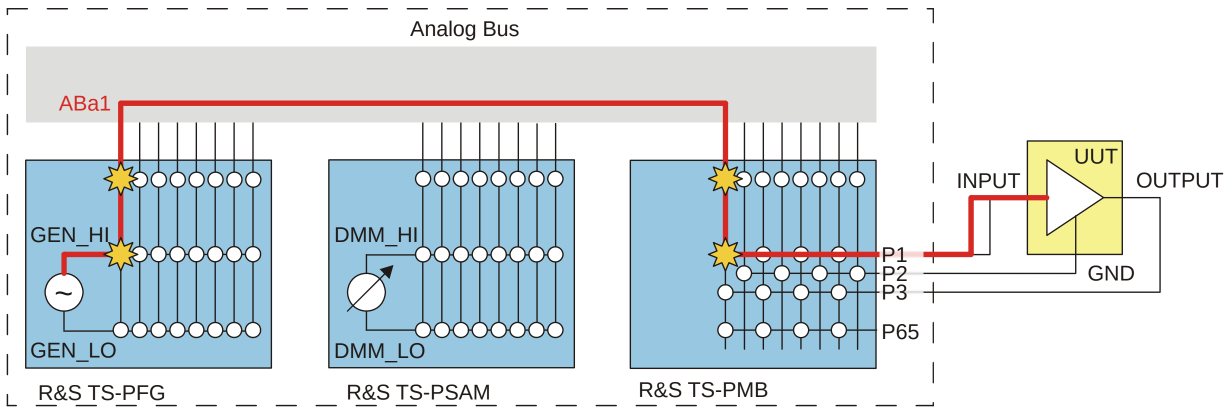


Figure 8-9 Ways to disconnect the GEN_HI > INPUT switched connection

The Signal Routing Library always attempts to disconnect the connection as close as possible to the UUT. In the example this means that the connection $LABa1 to INPUT will be opened on the matrix module. The remainder of the switched connection GEN_HI to $LABa1 of the R&S TS-PMB matrix module remains as an obsolete connection. The reference counter of an obsolete connection is 0, but the connection is still present physically.

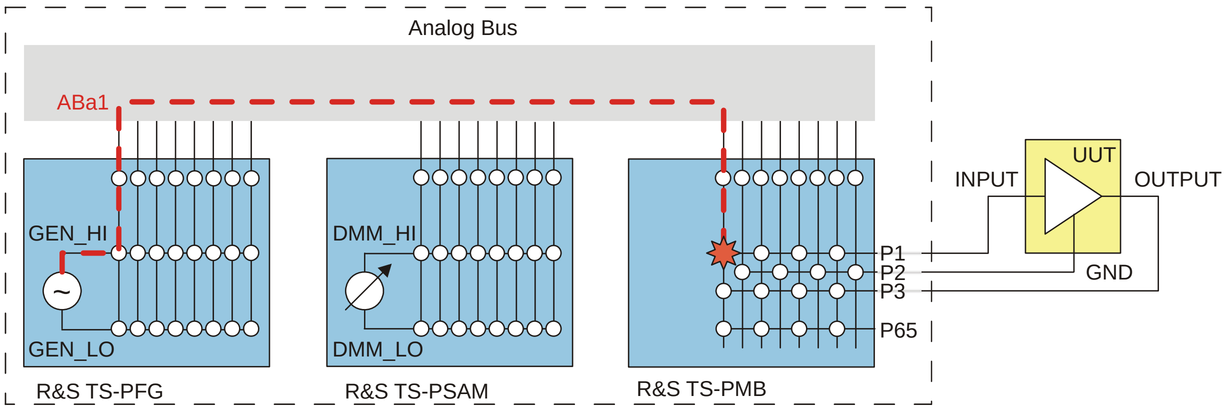


Figure 8-10 Obsolete connection GEN_HI > $LBABa1

In many cases, obsolete connections can be reused in subsequent switching commands. Example:

```
GEN_HI > MONITOR
```

Since the GEN_HI signal is already switched to $LABa1 of the R&S TS-PMB matrix module, it is sufficient to create the connection between $LABa1 and MONITOR by closing a single relay. In this manner obsolete connections can contribute to improved performance. This method extends the relay's life cycle, since switching processes are avoided.

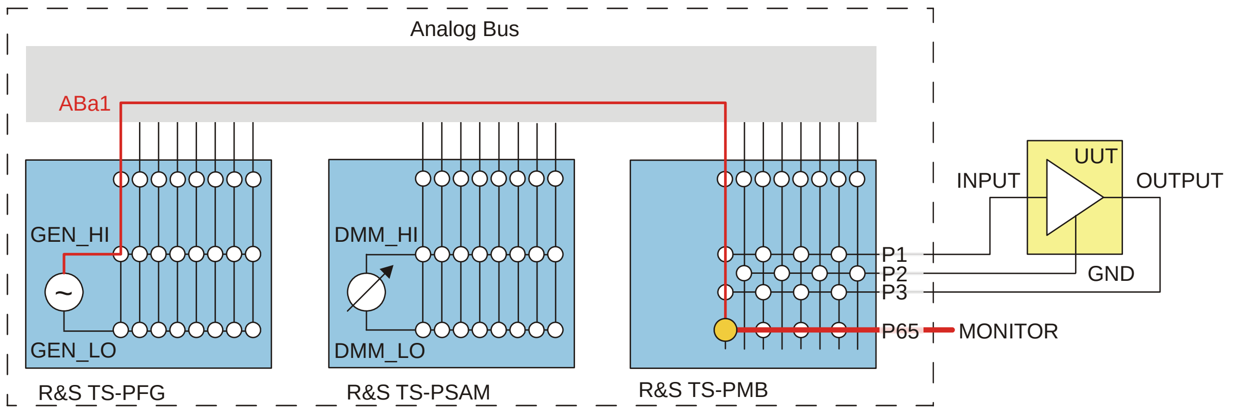


Figure 8-11 Reusing an obsolete connection

On the other hand, obsolete connections can block the analog measurement bus with signals that are no longer used. If the Signal Routing Library cannot find any more free analog measurement buses for automatic routing, it can release all obsolete connections by itself. That means that all obsolete connections are physically disconnected, i.e. the corresponding relays are opened. Only the active connections remain intact. The switching command “%” is used to selectively break all obsolete connections.

8.4.6.8 Analog measurement buses and coupling relays

All modules with analog measurement bus access have a local analog bus that can be connected with the global analog measurement bus. If a channel of a module needs to be connected with a global analog measurement bus, the coupling relays are automatically switched:

```
GEN_HI > $ABa1
```

This switching command first connects GEN_HI with the local analog measurement bus line LABa1 and then closes the coupling relay to connect LABa1 with ABa1.

```
GEN_HI || $ABa1
```

This switching command opens all connections between GEN_HI and ABa1, i.e. the relay that connects GEN_HI with the local analog bus line LABa1 is opened as well as the coupling relay between LABa1 and ABa1.

If only the coupling relay for analog measurement bus line ABa1 on the R&S TS-PFG module needs to be closed, the switching command should be as follows:

```
$LABa1 > $ABa1
```

This command is not possible because the assignment of the system name $LABa1 to the R&S TS-PFG module is not evident from the context. In this case the local analog measurement bus line must be explicitly added to the channel table:

```
[io_channel->test]
PFG.LABa1 = PFG!LABa1
```

The following switching commands closes the coupling relay for the analog measurement bus line ABa1 on module R&S TS-PFG:

```
PFG.LABa1 > $ABa1
```

The Signal Routing Library accepts the physical channel names "LABa1" to "LABd2" uniformly for all modules with analog measurement bus access. The same is true if the device driver of the module does not accept that channel name. This serves to standardise the differing treatment of coupling relays by device drivers in the Signal Routing Library.

8.4.6.9 Routing on R&S TS-PMB matrix modules

R&S TS-PMB matrix modules have two special features that must be taken into considerations for switched connections:

1. The even/odd rule applies to direct connections
2. Local analog buses can be connected in pairs via sense relays

The *even/odd rule* states that a matrix channel (P1 to P90) with an even number can only be connected to a local analog measurement bus that also has an even number. The same applies to channels and buses with odd numbers.

This rule does not apply to Instrument Lines IL1 to IL3. They can be connected with each local analog measurement bus.

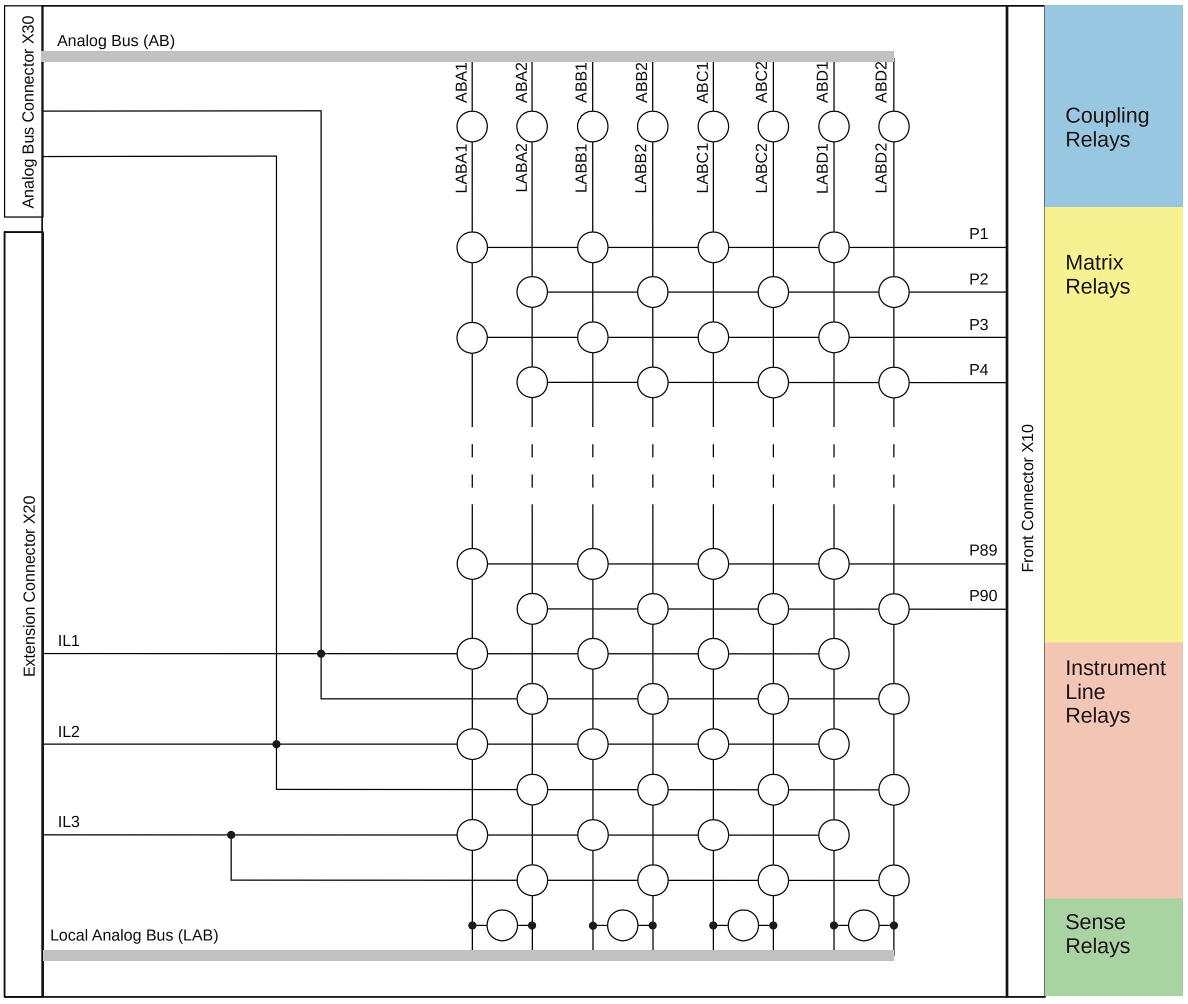


Figure 8-12 Block diagram R&S TS-PMB matrix

To make it possible nevertheless to connect an odd channel (for example P1) with an even local analog measurement bus (for example

LABa2), there is an option to connect an analog measurement bus pair with each other via a relay. The four so-called *sense relays* connect LABa1 - LABa2, LABb1 - LABb2, LABc1 - LABc2 and LABd1 - LABd2 respectively.

The following example illustrates the switching possibilities. The application channel table contains the following entries:

```
[io_channel->test]
P1      = PMB_10!P1
P2      = PMB_10!P2
P3      = PMB_10!P3
P4      = PMB_10!P4
; etc.
P89     = PMB_10!P89
P90     = PMB_10!P90
```

Channel P2 will be connected with global analog measurement bus ABa1:

```
P2 > $ABa1
```

To do this, P2 is first switched to LABa2. Then it is connected with LABa1 via the sense relay and is directed through the coupling relay to ABa1.

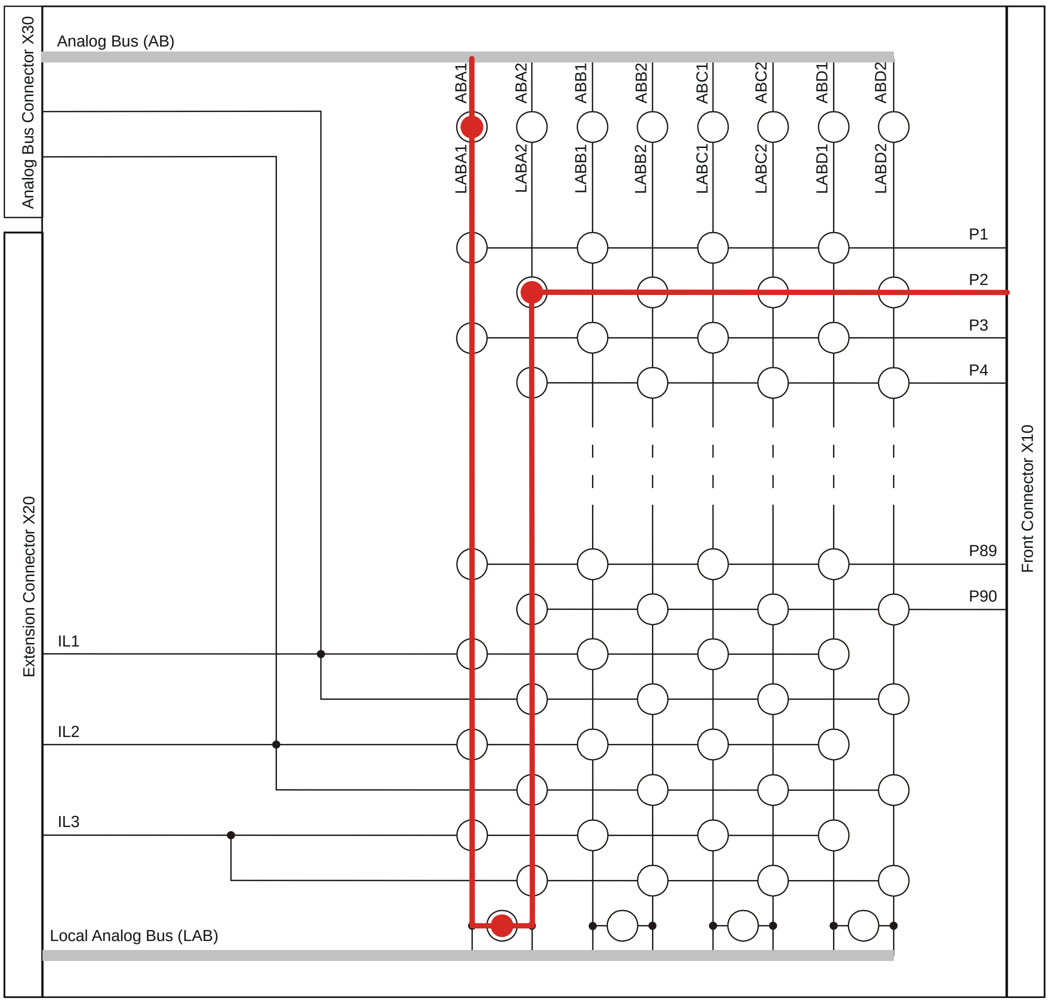


Figure 8-13 Switched connection P2 > $ABa1

The Signal Routing Library is capable of creating switched connections automatically via sense relays and coupling relays. A requirement for the example shown here is that other signals must not already be assigned to the two local analog measurement bus lines LABa1 and LABa2.

Local switched connections of channels to an R&S TS-PMB matrix module can be made in the same way:

```
P1 > P3, P4 > P89
```

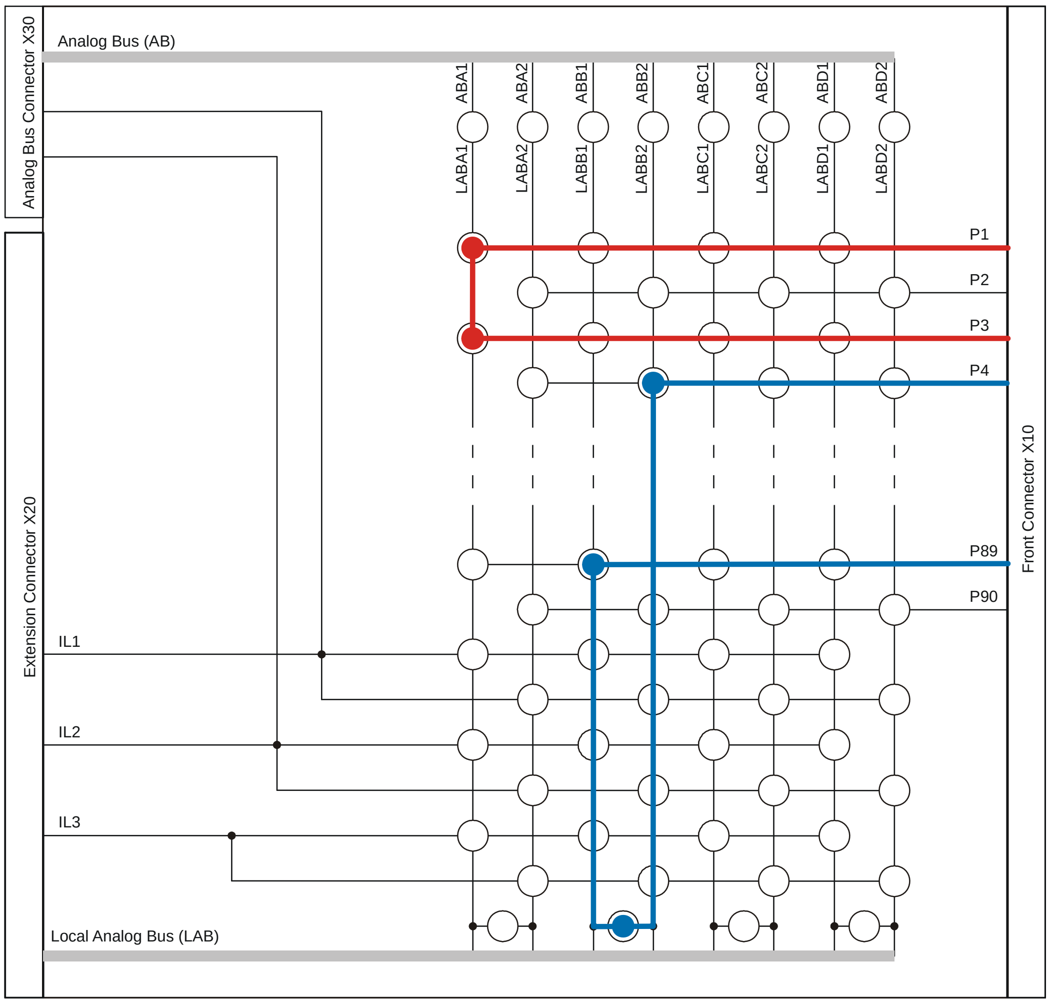


Figure 8-14 Switched connection P1 > P3, P4 > P89

The first connection leads from P1 through LABa1 to P3. Since both channels are odd, only a local analog measurement bus is required. An odd and an even channel are involved in the second connection. In this case the connection leads from P4 to LABb2, then via the sense relay to LABb1 and from there to P89.

8.4.6.10 Routing on power switching modules

A special rule related to local switching of channels applies to the power switching modules R&S TS-PSM1 and R&S TS-PSM2. If both channels are on the same module, only direct connections are permitted. Connections over the local analog measurement bus are not automatically routed.

The reason is that the local analog measurement bus is only suitable for currents up to maximum 1 A, but some channels of the power switching modules can switch up to 16 A. To prevent these high currents from being accidentally directed over the local analog measurement bus, automatic routing over the local analog measurement bus is not allowed for these modules.

However, this rule does not apply to connections between the switching module and the global analog measurement bus or other modules. Switched connections such as CH1_COM > $ABa1 or CH1_COM > DMM_HI are possible and are automatically directed via the coupling relays and the local and global analog measurement bus.

8.4.7 Using the Signal Routing Library with other libraries

The Signal Routing Library retains an image of the complete switched connections of the system in active memory. This makes it possible to calculate a new switched connection quickly, since no data transfer is required between the modules and the computer. This process does require, however, that no switched connections be performed outside the Signal Routing Library.

Because of this, the following calls are not permitted when using the Signal Routing Library:

- Calls to Switch Manager functions
- Direct calls to device drivers that change the switched connection
- Calls to functions in R&S GTSL libraries that switch the coupling relays
- Use of the automatic coupling relay mode with R&S TS-PMB modules.

CAUTION!

Calling functions that change the switching state of modules outside of the Signal Routing Library may result in faulty switched connections and short circuits. This may in turn damage the test system.

Special instructions must be followed

- When performing In-Circuit Tests (R&S EGTSL)
- When using the VACUUM Library
- When calling functions in device drivers or R&S GTSL libraries that reset modules.

8.4.7.1 Switch Manager

The Switch Manager Library SWMGR is the predecessor of the Signal Routing Library. The Switch Manager offers less functionality and is significantly more difficult to use. It can be used instead of the Signal Routing Library if compatibility with earlier R&S GTSL versions is required. The Signal Routing Library and Switch Manager are not designed for simultaneous use in a test program.

8.4.7.2 Device drivers

Functions in device drivers that affect the switched connections of modules must not be called if these modules are also administered by the Signal Routing Library, i.e. if they are configured as "SwitchDevice<i>". Examples of these functions are:

- *prefix*`_Connect`
- *prefix*`_Disconnect`
- *prefix*`_DisconnectAll`
- *prefix*`_SetPath`

The placeholder *prefix* stands for the function prefix of the device driver, for example rspsam or rspmb. Some device drivers offer additional switching functions, for example `rspsam_cnx_Matrix`, which also must not be used. Functions that affect the ground reference of a device, such as `rspsam_cnx_Gnd` or `rspfg_ConfigureGround`, are exceptions to this rule.

8.4.7.3 Coupling relays

Functions in R&S GTSL libraries that affect switched connections must not be called. These functions are:

- `DMM_Conf_Coupling_Relays` (DMM.DLL)
- `FUNCGEN_ConfigureCouplingRelays` (FUNCGEN.DLL)
- `SIGANL_ConfigureCouplingRelays` (SIGANL.DLL)

Device driver functions that affect switched connections of coupling relays must not be used.

Note that with R&S TS-PMB modules the automatic mode of the coupling relays must not be used together with the Signal Routing Library. Therefore the RSPSAM_ATTR_CR_AUTO attribute must not be set to VI_TRUE. Use of “DriverSetup=CRAuto:1” in the DriverOption entry of PHYSICAL.INI is also not permitted. The Signal Routing Library ensures that the setting of the coupling relay mode is correct in the `ROUTE_Setup` function.

8.4.7.4 In-Circuit Test

The In-Circuit Test Library ICT.DLL may be used together with the Signal Routing Library if the following rules are observed:

Function **`ICT_Load_Program`** prepares the modules entered as ICTDevice<i> and the R&S TS-PMB matrix modules entered as SwitchDevice<i> for use in the In-Circuit Test. At the same time the modules are reset. This function may only be called if the Signal Routing Library has not switched any connections.

The functions **`ICT_Run_Program`** and **`ICT_Debug_Program`** also reset the modules and change the coupling relays. When the In-Circuit Test is being performed, make certain that no signals are switched to the global analog measurement bus lines.

CAUTION!

Before the ICT functions named above are called, all active and obsolete connections of the Signal Routing Library must be opened with the switching command “||” (Disconnect All). Signals on global analog measurement bus lines when an In-Circuit Test is called may result in incorrect measurements, faulty switched connections or short circuits. This may in turn damage the test system.

If the ICT program is terminated and these functions return, the configured modules (ICTDevice<i> and R&S TS-PMB matrix modules entered as SwitchDevice<i>) are in reset state.

The function **`ICT_Unload_Program`** also resets configured modules to the basic state. They may only be called if the Signal Routing Library has not switched any connections.

When using **`ICT Extension Libraries`** the same instructions apply for the modules that are used by those libraries.

8.4.7.5 VACUUM Library

The Vacuum Library VACUUM.DLL can be used together with the Signal Routing Library if the R&S TS-PSYS1 or R&S TS-PSYS2 modules configured as VacuumControl<i> are not also configured as SwitchDevice<i>. If these rules are not observed the switching command “||” causes the vacuum to be deactivated.

8.4.7.6 Reset modules

Functions that trigger a reset of modules must only be called if it is assured that the Signal Routing Library on these modules has no more active or obsolete connections. It is recommended to perform the switching command “||” (Disconnect All) previously.

8.4.8 Panel test

A *panel test* is a test of a number of identical test units, called UUTs or units under test, which are arranged on a panel, i.e. a carrier shared by the units. The UUTs are tested either one after the other or simultaneously.

Therefore the test programs for the individual UUTs on the panel differ only in adapter wiring, not in sequence. This makes it efficient to create a single test program that is used for all UUTs and to control the variable switched connection with configuration files. This procedure minimises the effort required to maintain the test program, because each change affects all UUTs simultaneously.

A separate bench in APPLICATION.INI and a separate application-specific channel table has to be created for each UUT on the panel. The logical channel names are the same, but the assignment to physical channel names differs according to adapter wiring.

In the following example, differences between the two benches are highlighted in colour:

```
[bench->test1]

DigitalMultimeter = device->PSAM
FunctionGenerator = device->PFG
SwitchDevice1     = device->PSAM
SwitchDevice2     = device->PMB_10
SwitchDevice3     = device->PFG

AnalogBus         = device->ABUS

AppChannelTable   = io_channel->test1

[io_channel->test1]
INPUT             = PMB_10!P1
GND               = PMB_10!P2
OUTPUT            = PMB_10!P3
MONITOR           = PMB_10!P65
```

```
[bench->test2]

DigitalMultimeter = device->PSAM
FunctionGenerator = device->PFG
SwitchDevice1     = device->PSAM
SwitchDevice2     = device->PMB_10
SwitchDevice3     = device->PFG
SwitchDevice4     = device->PMB_14
AnalogBus         = device->ABUS

AppChannelTable   = io_channel->test2

[io_channel->test2]
INPUT             = PMB_14!P33
GND               = PMB_14!P34
OUTPUT            = PMB_14!P35
MONITOR           = PMB_10!P65
```

Devices used in common such as PSAM and PFG and PMB_10 occur in both benches. Bench “test2” also uses matrix module PMB_14.

The logical channel names in the two channel tables are identical. Physical channel names differ depending on different adapter wiring. The MONITOR channel is the same for both channel tables. In this case, an oscilloscope will be connected to make it possible to monitor the test signals of both UUTs.

NOTE:

The even/odd relationship of channels should be retained in all channel tables for the panel test. This means either only even or only odd physical channels should be assigned to a logical channel on an R&S TS-PMB matrix module.

The program sequence for the panel test is such that the `ROUTE_Setup` function is is called once at the beginning for each UUT passing the appropriate bench name. This results in a separate resource ID for the Signal Routing Library for each UUT.

The corresponding resource ID for the each UUT is passed in all subsequent switching functions.

Example:

```
#define NUM_UUTS   2

short errorOccurred;
long  errorCode
char  errorMessage[GTSL_ERROR_BUFFER_SIZE];
long  resourceId[NUM_UUTS];

// setup libraries
RESMGR_Setup ( 0, "physical.ini", "testApplication.ini",
               &errorOccurred, &errorCode, errorMessage  );

// setup library for each UUT in the panel
for ( i = 0; i < NUM_UUTS; i++ )
{
  char benchName[80];
  sprintf ( benchName, "bench->test%d", i+1  );
  ROUTE_Setup ( 0, benchName, &resourceId[i],
                &errorOccurred, &errorCode, errorMessage  );
}

// call tests for each UUT
for ( i = 0; i < NUM_UUTS; i++ )
{
  // connect generator
  ROUTE_Execute ( 0, resourceId[i], "GEN_LO > GND,  GEN_HI > INPUT",
                  &errorOccurred, &errorCode, errorMessage  );

  // connect DMM
  ROUTE_Execute ( 0, resourceId[i], "GND > DMM_LO,  OUTPUT > DMM_HI",
                  &errorOccurred, &errorCode, errorMessage  );

  // disconnect all
  ROUTE_Execute ( 0, resourceId[i], "||",
                  &errorOccurred, &errorCode,  errorMessage );
}

// close library for each UUT
for ( i = 0; i < NUM_UUTS; i++ )
{
  ROUTE_Cleanup ( 0, resourceId[i], &errorOccurred, &errorCode,
                  errorMessage );
}

RESMGR_Cleanup ( 0, &errorOccurred, &errorCode,  errorMessage);
```

In the example above, the number of UUTs is defined as a constant NUM_UUTS. The resource IDs for the individual UUTs are kept in an array resourceId[NUM_UUTS] The relevant UUT is selected by means of the index variable i. In this manner the program can easily be adapted to another number of UUTs on the panel by defining NUM_UUTS appropriately. In addition, the appropriate number of benches and channel tables must also be added to APPLICATION.INI.

Switch settings can also be used in the panel test. Since they contain only logical channel names it is sufficient to create a table used in common with switch settings and to add a reference to the table to each bench.

All switching commands affect only the SwitchDevice<i> configured in the relevant bench by the resource ID, with the following exception:

NOTE:

The two commands “||” and “%” affect all hardware elements, i.e. all switch modules that are configured in any bench as Switch-Device<i>.

8.4.9 Error cases

If an error occurs in a switching command, the command is aborted and the error is reported. In addition, the switching state that was present before the `ROUTE_Execute` function was called is restored.

Some switching commands cannot be undone once they are complete. These are “||” (Disconnect All) and “%” (Disconnect all obsolete connections). Because of this it is not guaranteed that all connections will still be in existence after an error.

If a warning occurs during a switching command, the execution of the command continues and the warning is reported. If more than one warning occurs in a command, only the first warning is reported.

8.4.10 Integrating third-party modules

In addition to R&S CompactTSVP modules, the Signal Routing Library can also control third-party switch modules. A requirement in this case is that the modules provide a IVI-C driver of class *IviSwtch*. A section with the following information must be entered in PHYSICAL.INI for each external module:

Keyword	Value	Description
Type	String	*Mandatory entry* IVI_SWITCH
ResourceDesc	String	*Mandatory entry* Resource descriptor string
DriverDll	String	*Mandatory entry* File name of the device driver DLL
DriverPrefix	String	*Mandatory entry* Prefix for the IVI driver functions
DriverSetup	String	*Optional entry* Optional indications which are passed to the device driver during the *prefix*_InitWithOptions function.

Table 8-9 Entries in PHYSICAL.INI for third-party modules

Refer to the documentation for the module about the entries required for ResourceDesc, DriverDll, DriverPrefix and DriverSetup.

Sample entry for an NI2565 switch module:

```
[device->NI2565]
Type         = IVI_SWITCH
ResourceDesc = PXI1::15::INSTR
DriverDLL    = niswitch_32.dll
DriverPrefix = niSwitch
```

8.4.11 Examples

8.4.11.1 Scanner

In the following example, three voltages will be measured between two channels HI and LO. To do this, channels DMM_HI and DMM_LO are connected with the corresponding UUT channels and the voltage is measured. Then the connections are disconnected.

```
[io_channel->scanner]
DMM_HI = psam!dmm_hi
DMM_LO = psam!dmm_lo
HI1    = pmb_10!P1
HI2    = pmb_10!P2
HI3    = pmb_10!P3
LO1    = pmb_10!P4
LO2    = pmb_10!P5
LO3    = pmb_10!P6
```

The corresponding switching commands are:

```
DMM_HI > HI1, DMM_LO > LO1,  ?#
```

(Perform measurement)

```
DMM_HI | HI1, DMM_LO | LO1,  ?#
```

Then the same process is performed for channel pairs HI2 / LO2 and HI3 / LO3.

```
DMM_HI > HI2, DMM_LO > LO2, ?#
DMM_HI | HI2, DMM_LO | LO2, ?#
DMM_HI > HI3, DMM_LO > LO3, ?#
DMM_HI | HI3, DMM_LO | LO3, ?#
```

For the first measurement, DMM_HI through $ABa1 is connected with HI1 (odd channel) and DMM_LO through $ABb2 is connected with LO1 (even channel):

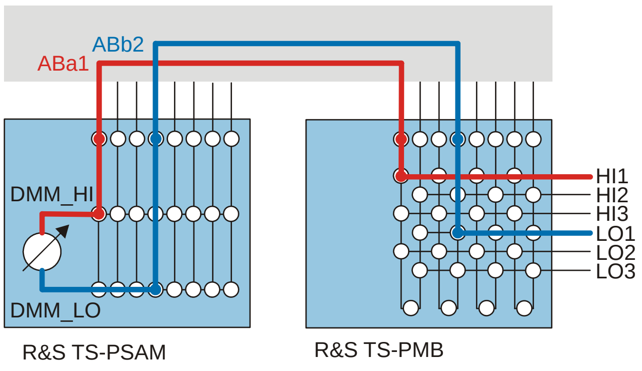


Figure 8-15 Scanner, measurement on HI1 and LO1

HI2 (even channel) and LO2 (odd channel) are used for the second measurement. Now a direct connection from $ABa1 to HI2 is no longer possible. The connection to HI2 is now made via the sense relay or the R&S TS-PMB module and the local analog measurement buses $LABa1 and $LABa2. The connection to LO2 is also made by a sense relay.

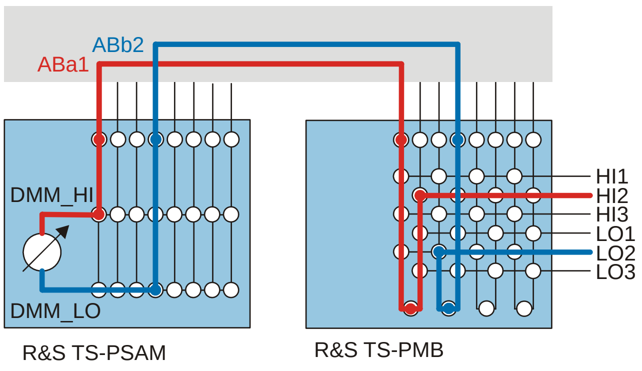


Figure 8-16 Scanner, measurement to HI2 and LO2 via sense relays

In the following measurement on HI3 and LO3 both channels can again be connected directly, i.e. without sense relays, with the analog measurement buses. The connections which are still set up through the sense relays remain in existence as obsolete connections.

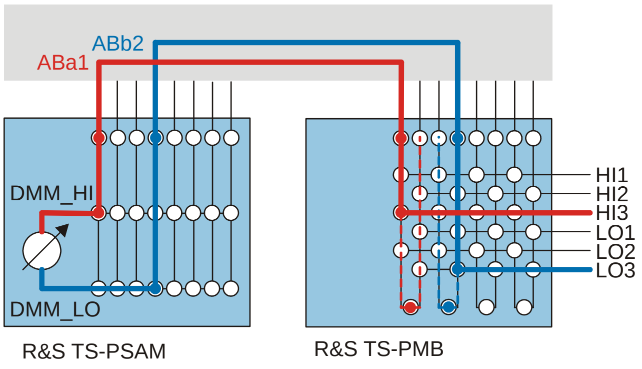
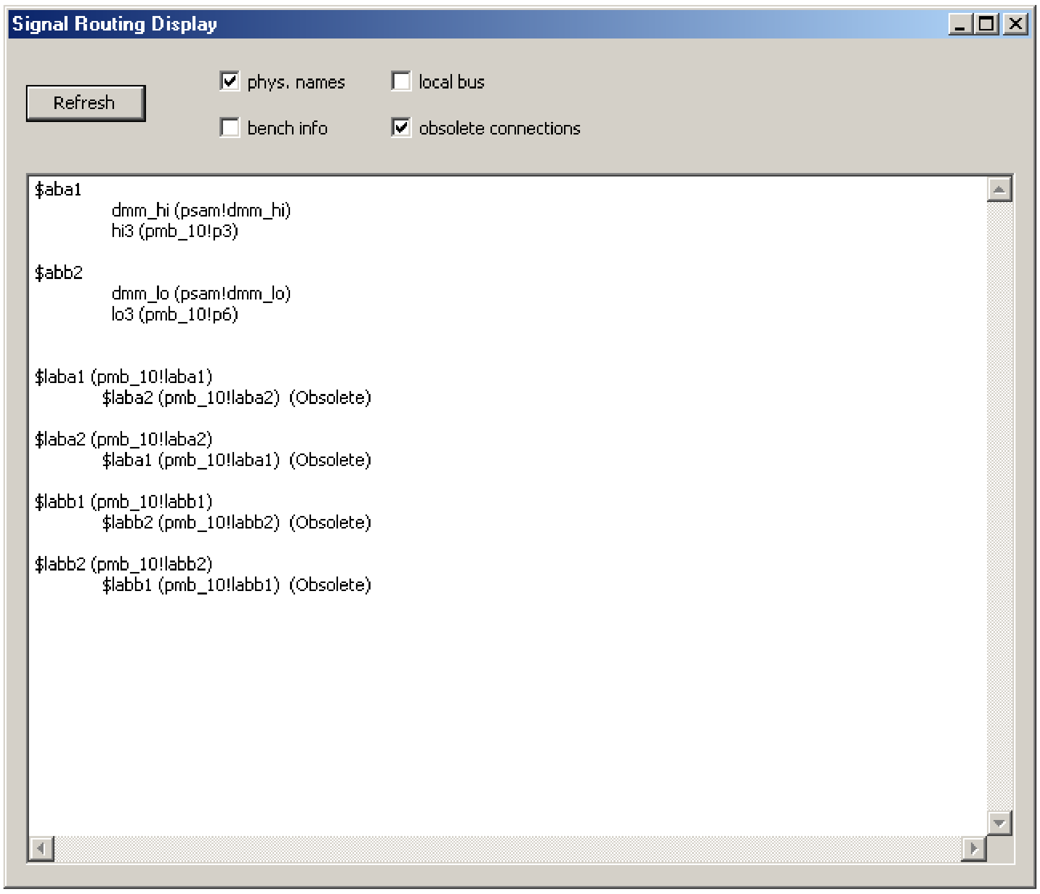


Figure 8-17 Scanner, measurement on HI3 and LO3 with obsolete connections.

Active and obsolete connections can be seen in the switched connection display:



[✓] phys. names
[ ] local bus
[ ] bench info
[✓] obsolete connections

```
$aba1
            dmm_hi (psam!dmm_hi)
            hi3 (pmb_10!p3)

$abb2
            dmm_lo (psam!dmm_lo)
            lo3 (pmb_10!p6)


$laba1 (pmb_10!laba1)
          $laba2 (pmb_10!laba2)  (Obsolete)

$laba2 (pmb_10!laba2)
          $laba1 (pmb_10!laba1)  (Obsolete)

$labb1 (pmb_10!labb1)
          $labb2 (pmb_10!labb2)  (Obsolete)

$labb2 (pmb_10!labb2)
          $labb1 (pmb_10!labb1)  (Obsolete)
```

Figure 8-18 Switched connection display for measurement on HI3 and LO3

8.4.11.2 Current measurement via shunt

Power switching module R&S TS-PSM1 offers the possibility of measuring high currents with a voltage measurement on the integrated shunt resistors.

When setting up the connections for this measurement, note that the channel name is the same *on both sides* of the shunt resistor, for example CH1no for channel 1. To ensure that DMM_HI and DMM_LO are directed to the two different connections of the shunt resistor, however, the analog measurement bus lines must be explicitly specified:

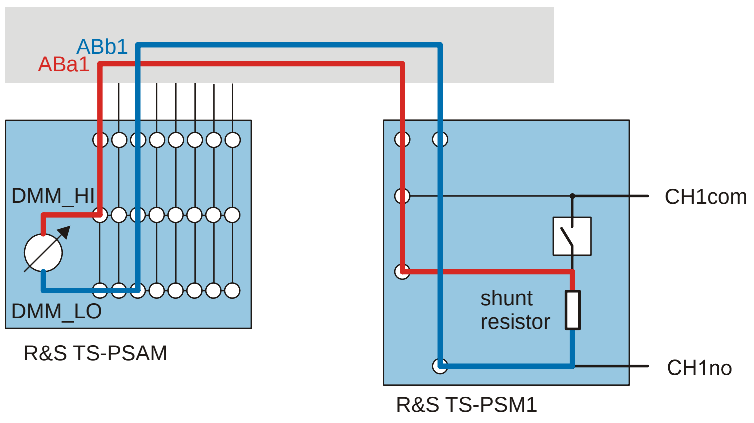


Figure 8-19 Current measurement via R&S TS-PSM1 shunt resistor

The lower connection of the shunt resistor in Figure 8-19 can only be reached through analog measurement bus ABb1. The upper connection can only be reached through ABa1. The switching command is therefore as follows:

```
DMM_HI > $ABa1 > CH1no
DMM_LO > $ABb1 > CH1no
```

9 Creation of test libraries

NOTE:

Knowledge of C programming is needed to create test libraries. Do not modify the original Sample Library project. Instead, make a copy of the Sample directory and make your modifications in the copy.
Be careful when you put your private DLLs in the `gtsl\bin` directory. Always keep a copy, because they may be overwritten by an update to R&S GTSL if there is a DLL with the same name. It is safer to keep your projects and DLLs in a completely separate location beside the R&S GTSL tree. Be sure to add the directory where your DLLs reside to the PATH environment variable of your computer.

9.1 Scope

9.1.1 Identification

This chapter describes how to write a high-level library for the R&S GTSL software.

9.1.2 System overview

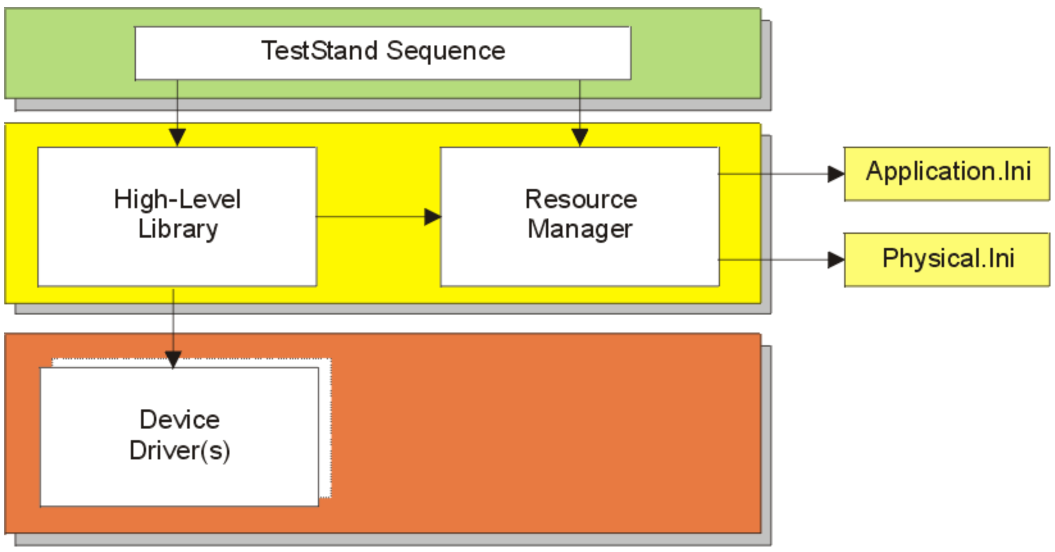


Figure 9-1 R&S GTSL software overview

A high-level library offers a group of functions which cover the needs for testing a specific class of UUTs. As an example, the GSM library includes all functions required for final and functional tests of GSM mobile phones.

The library functions are called from a TestStand sequence. The functions themselves interact with the resource manager library and the device driver(s).

9.2 Referenced documents

[INSTR]	LabWindows/CVI Instrument Driver Developers Guide, National Instruments, February 1998 Edition
[RESMGR]	Resource Manager online help file (resmgr.hlp)

9.3 Software design decisions

The design of any R&S GTSL high-level library must meet the following requirements:

- The library must be delivered as a Dynamic Link Library (DLL), including type library information, a function panel and a Windows help file.
- The function call interface must follow the R&S GTSL template.
- Each library must offer a Setup and a Cleanup function as well as a Lib_Version function.
- The library must use the resource manager functions (if applicable).
- The library must be thread safe.
- The library must support device sharing using the lock/unlock functions of the resource manager.

These requirements are described in detail in the following sections.

9.4 Architectural design

9.4.1 Components

The basic components of each high-level library are:

- High-level Library 'xyz': (deliverable items)

– xyz.h	include file
– xyz.lib	import library
– xyz.dll	dynamic link library
– xyz.fp	CVI function panel
– xyz.hlp	Windows help file

- High-level Library 'xyz' (non-deliverable items)

– xyz.c	source file
– xyz.prj	CVI project file
– ...	additional source/include files (if applicable)

9.4.2 Concept of execution

The high-level library functions are called from the TestStand sequencer using the DLL flexible prototype adapter. The function prototypes follow the TestStand default template for the DLL flexible prototype adapter and allow easy integration into the TestStand environment.

Each high-level library offers a setup function, which identifies the device(s) and options to be used, initializes the appropriate device drivers and puts the device(s) into a proper operating state. The setup function must be called from the TestStand sequence before any other library function. The cleanup function must be called at the end of all tests, it releases the devices and frees the resources. The Lib_Version function returns the version string of the library.

The setup function returns a resource id, i.e. a unique identifier, which must be used for all subsequent function calls of the high-level library.

A set of functions is provided depending on the purpose of the library. The GSM library, for example, offers configuration and measurement functions for signaling and non-signaling mode as well as audio measurement functions. Each function can be seen as a single test step (or test case) in the TestStand sequence. Therefore, it should execute a single operation (like a measurement) and return a single value (like a power level, a bit error rate, etc...). In this case, the measured value can be compared and logged in a single test step in the TestStand sequence, keeping the sequence short and readable. Some measurement functions may return more than one value (like an amplitude and a phase) if necessary. In this case, an additional comparison step is required in the TestStand sequence.

The library functions should be 'high-level', which means not too simple (requiring a lot of steps in the sequence to set up a device or to take a measurement) and not too complicated (returning a bunch of measured values which must be stored by the TestStand sequence and validated by several subsequent steps).

The functions are based on the functionality of the underlying device driver. Depending on the complexity of the function and the features of the device driver, this may be a simple call of a single device driver function or a complex piece of code dealing with several device drivers and a large number of driver calls.

There are a number of benefits to using a high-level library instead of calling a device driver directly:

- the library can handle more than one device (bench concept)
- the library can switch between different types of devices without modification of the TestStand sequence (e.g. GSM library: substitution of a CMD55 with a CMU)
- the library implements device sharing between several processes or threads (locking)
- standard INI-file concept for resource description (physical/application layer)
- standard error handling mechanism, suited for use with TestStand

The required functionality is provided by the Resource Manager library. The resource manager is one of the central parts of the R&S GTSL software. It coordinates the interaction between all the libraries, especially in parallel test scenarios. Therefore it is mandatory to include the resource manager in each high-level library.

9.4.3 Interface design

9.4.3.1 Interface identification and diagrams

A high-level library has the following interfaces:

- Export functions (interface to TestStand sequence)
- Driver calls (interface to device drivers)
- Resource Manager calls (interface to resource manager)
- Resource Description (physical/application layer, via resource manager)
- Function Panel User Interface

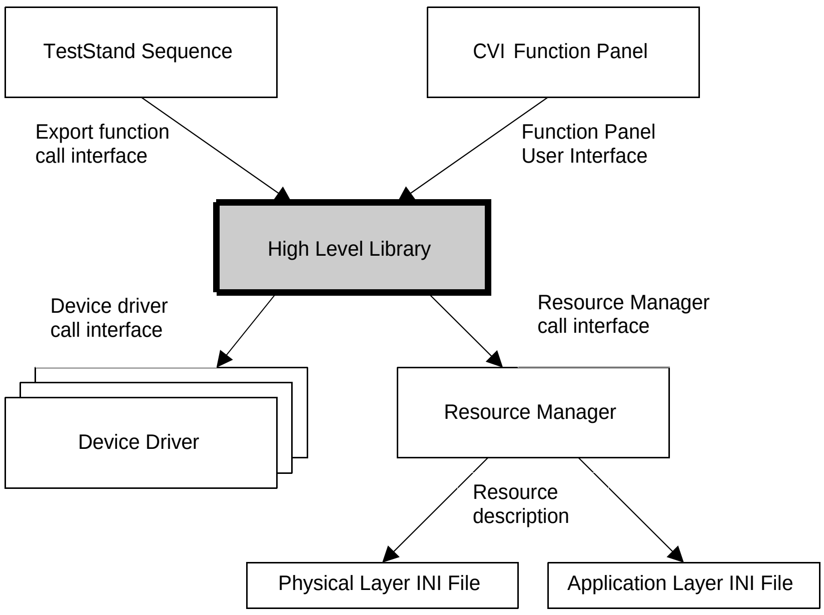

TestStand Sequence
CVI Function Panel
Export function call interface
Function Panel User Interface
High Level Library
Device driver call interface
Resource Manager call interface
Device Driver
Resource Manager
Resource description
Physical Layer INI File
Application Layer INI File

Figure 9-2 Interface diagram

9.4.3.2 Export functions

The export function prototype follows the default template for the DLL flexible prototype adapter, which is defined by National Instruments in the TestStand software. This prototype allows easy integration of the export functions into the TestStand environment, especially for automatic error handling.

The R&S GTSL template follows the default template, but is more flexible in terms of the number and types of the parameters.

9.4.3.2.1 Naming conventions

Each function of a library begins with the library prefix followed by an underscore character. The prefix is normally the same as the name of the library and must be written in uppercase characters. For example, all function names of the GSM library begin with 'GSM_'.

The function name shall clearly show the action behind it. If the functions are arranged in groups, the group shall also be part of the function name. For example, the name for a measurement function for the peak power in non-signaling mode in the GSM library may be 'GSM_NonSig_Meas_Power_PK'. In this case, 'GSM_' is the library prefix, 'NonSig_' is a function group for non-signaling mode, 'Meas_' is a sub-group for measurement functions, and 'Power_PK' is the parameter to be measured. The function name is written in mixed uppercase/lowercase with underscores to make it more readable.

9.4.3.2.2 Function type and calling convention

The function type is void DLLEXPORT DLLSTDCALL, i.e. there is no return value. The function is exported to the export library (DLLEXPORT) and the call interface is the standard call interface (DLLSTDCALL). The DLLEXPORT and DLLSTDCALL defines are taken from the cvidef.h include file and are compiler independent, while the constructs ”__declspec(dllexport)” and ”__stdcall” may be a problem when the compiler is not LabWindows/CVI or Microsoft Visual C.

9.4.3.2.3 Function parameters

sequenceContext	The first parameter is always 'CAObjHandle sequenceContext'. When a library function is called from TestStand, the built-in variable 'ThisContext' is passed. When the library is used outside TestStand, this parameter must be set to zero. The sequence context allows access to the TestStand sequence and variables from within the library using the ActiveX properties and methods of TestStand. Normally, a high-level library will not use the sequence context. However, it must be passed to the resource manager functions as a parameter.
resourceId	A number which identifies the resource, i.e. the bench or device.

The second parameter is the resource ID, which is obtained when the library setup function is called. The resource ID must be stored in the TestStand sequence after the call of the setup function, and it must be passed to all subsequent function calls into the library.

...	Additional parameters

Additional parameters like setup parameters and pointers to result values are inserted in the second, third,... place as required. Input parameters should precede output parameters.

pErrorOccurred	output parameter, passed by reference Error flag, is set to 1 if an error occurred, otherwise 0.
pErrorCode	output parameter, passed by reference < 0 : Error code in case of pErrorOccurred = 1 = 0 : Function returned successfully, pErrorOccurred = 0 > 0 : Indicates a warning when pErrorOccurred = 0.
errorMessage	output parameter The error message will be copied to this buffer in case of an error or warning, the contents remain unchanged if the function completes successfully. The minimum size of this buffer must be GTSL_ERROR_BUFFER_SIZE (1024 bytes).

The last three parameters are used for error handling in TestStand. They are defined as 'short *pErrorOccurred', 'long *pErrorCode' and 'char errorMessage[]'.

9.4.3.2.4 Example

The following example shows the prototype of a function for a capacitor measurement. There are three additional input parameters (measMode, stimVoltage and stimFrequency) and two output paramters (capacitance and phase).

```
void DLLEXPORT DLLSTDCALL SAMPLE_Meas_Capacitance
      ( CAObjHandle   sequenceContext,
        long          resourceId
        int           measMode,
        double        stimVoltage,
        double        stimFrequency,
        double      * capacitance,
        int         * phase
        short       * pErrorOccurred,
        long        * pErrorCode,
        char          errorMessage[]);
```

9.4.3.2.5 Setup function

The setup function is mandatory. The name of the setup function is defined as the library prefix with underscore followed by the word 'Setup'. The parameter list is shown below. ResourceName (input parameter) is a character string which identifies the resource(s) in the application and physical layer INI files. This may be a logical name, the name of a bench or the name of a device.

ResourceID is an output parameter. If the function returns successfully, a resource ID for the allocated resources is returned here.

```
void DLLEXPORT DLLSTDCALL SAMPLE_Setup
      ( CAObjHandle   sequenceContext,
        char        * resourceName,
        long        * resourceId,
        short       * pErrorOccurred,
        long        * pErrorCode,
        char          errorMessage[] );
```

9.4.3.2.6 Cleanup function

The cleanup function is mandatory. The name of the cleanup function is defined as the library prefix with underscore followed by the word 'Cleanup'. The parameter list is described in section 9.4.3.2.3, there are no additional parameters.

```
void DLLEXPORT DLLSTDCALL SAMPLE_Cleanup
      ( CAObjHandle   sequenceContext,
        long          resourceId
        short       * pErrorOccurred,
        long        * pErrorCode,
        char          errorMessage[]);
```

9.4.3.2.7 Library version function

The library version function is mandatory. The name of the library version function is defined as the library prefix with underscore followed by the words 'Lib_Version'. This function may be called any time, i.e. even before calling the setup function. Therefore, there is no resourceId parameter to this function. The version string is copied to the string buffer libraryVersion.

```
void DLLEXPORT DLLSTDCALL RESMGR_Lib_Version
     ( CAObjHandle   sequenceContext,
       char          libraryVersion[80],
       short       * pErrorOccurred,
       long        * pErrorCode,
       char          errorMessage[] );
```

9.4.3.3 Device driver interface

The interface to the device driver is mainly dependent on the device driver itself. A device driver may be conforming to

- the IVI standard
- the VISA standard
- none of the above

and it may be

- written by Rohde & Schwarz
- written by a third party developer

The Resource Manager supports device drivers using the VISA and IVI standards (the session functions support a data type ViSession, which is the session handle to a VISA session). Non-standard drivers may be supported as long as there is something comparable to the ViSession handle (like a file pointer or handle for a serial interface in the Windows API, as long as its size does not exceed 32 bits).

A high-level library may deal with more than one device driver in two cases:

- The library supports several devices as an alternative (e.g. CMD55 or CMU).
- The library requires several devices concurrently to perform a task (e.g. FSE and SMIQ).

The Setup function of the high-level library must call the Initialize function of the device(s) and establish a connection to the hardware. It may also be necessary to setup the device to a known state.

The Cleanup function of the high-level library must call the Close function of the device driver, closing the connection to the hardware. It may be necessary to reset the device to a known state before.

Both functions must implement the cooperative session handle concept (see 8.5.2.2ff). This means that the Setup function must not initialize the driver if a session already exists, and the Cleanup function must not close the driver if there is still another session open to the device. This concept is important in parallel test scenarios with device sharing.

9.4.3.4 Resource Manager interface

The Resource Manager interface is described in the RESMGR.HLP file.

How the interface is to be used, will be described in the following sections of this document. The SAMPLE project includes the mandatory functions of a high-level library (Setup, Cleanup, Lib_Version) as well as one measurement function. These functions show how to use the Resource Manager interface.

9.4.3.5 Resource Description

The syntax of the Resource Description is defined in Section 5. The resources are described in two INI-file style text files, the physical layer INI-file and the application layer INI-file.

The physical layer contains physical device information like the device type and the IEEE address:

```
[device->CMD55]
Description = Radio Communication Tester CMD55
Type = CMD55
ResourceDesc = GPIB0::15
```

The application layer contains application-specific information, which may vary for different UUTs:

```
[LogicalNames]
GSM = bench->Radiocom_GSM

[bench->Radiocom_GSM]
Description = Bench for GSM library
RadioComTester = device->CMD55
Simulation=0
```

The Setup function of a high-level library reads the configuration information and determines the hardware which is actually present. The meaning of these entries is described in detail in section 9.5.3.

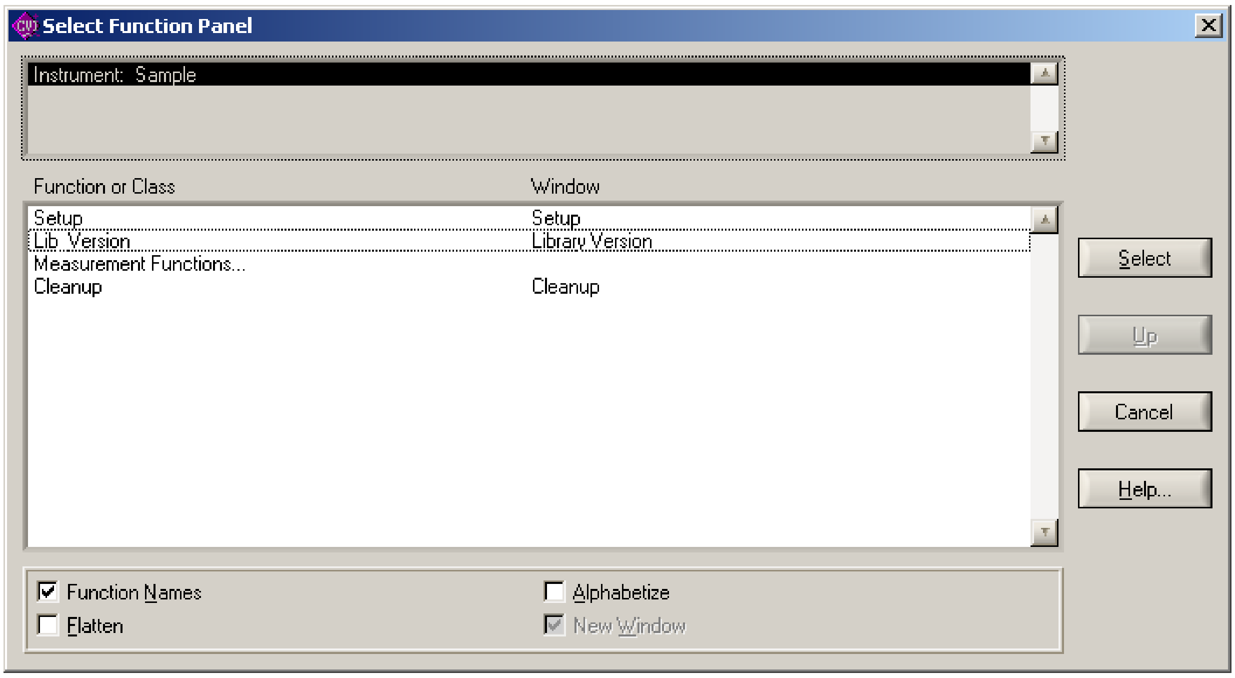
9.4.3.6 Function Panel User Interface

The Function Panel User Interface is an interactive graphical interface that assists the software developer in understanding what each particular library function does and how to use the programmatic developer interface to call each function.

See the “National Instruments documentation” for details on the LabWindows/CVI function panels, the Function Tree Editor and the

Function Panel Editor.

The function tree contains the three standard functions Setup, Lib_Version and Cleanup. The other functions are grouped into sub-trees (like Measurement, Configuration, Signaling, Non-signaling etc.).



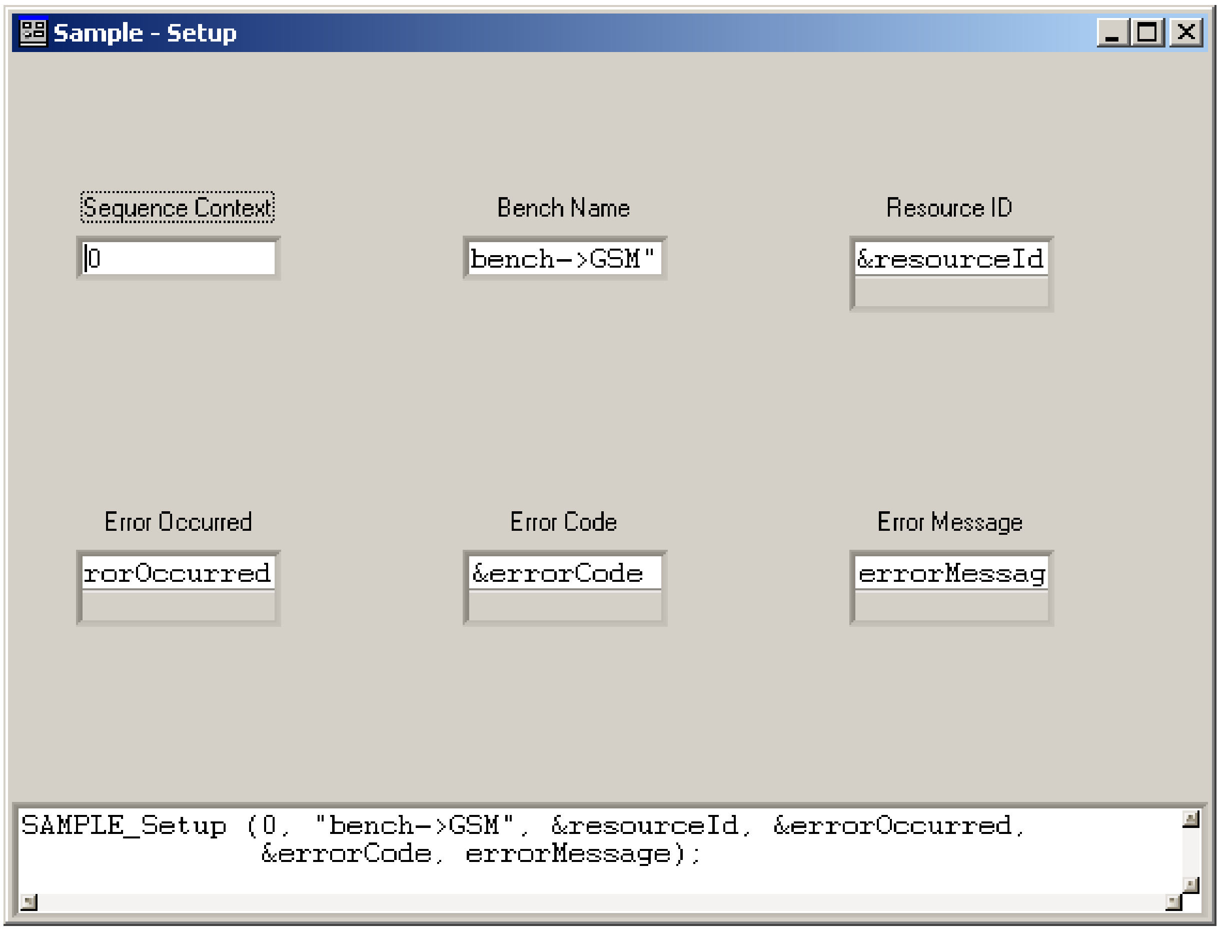
Function or Class	Window
Setup	Setup
Lib_Version	Library Version
Measurement Functions...	
Cleanup	Cleanup

☑ Function Names
☐ Alphabetize
☐ Flatten
☑ New Window

Figure 9-3 Function tree of the SAMPLE project

There is a function panel window for each function:

17th Issue 08.09

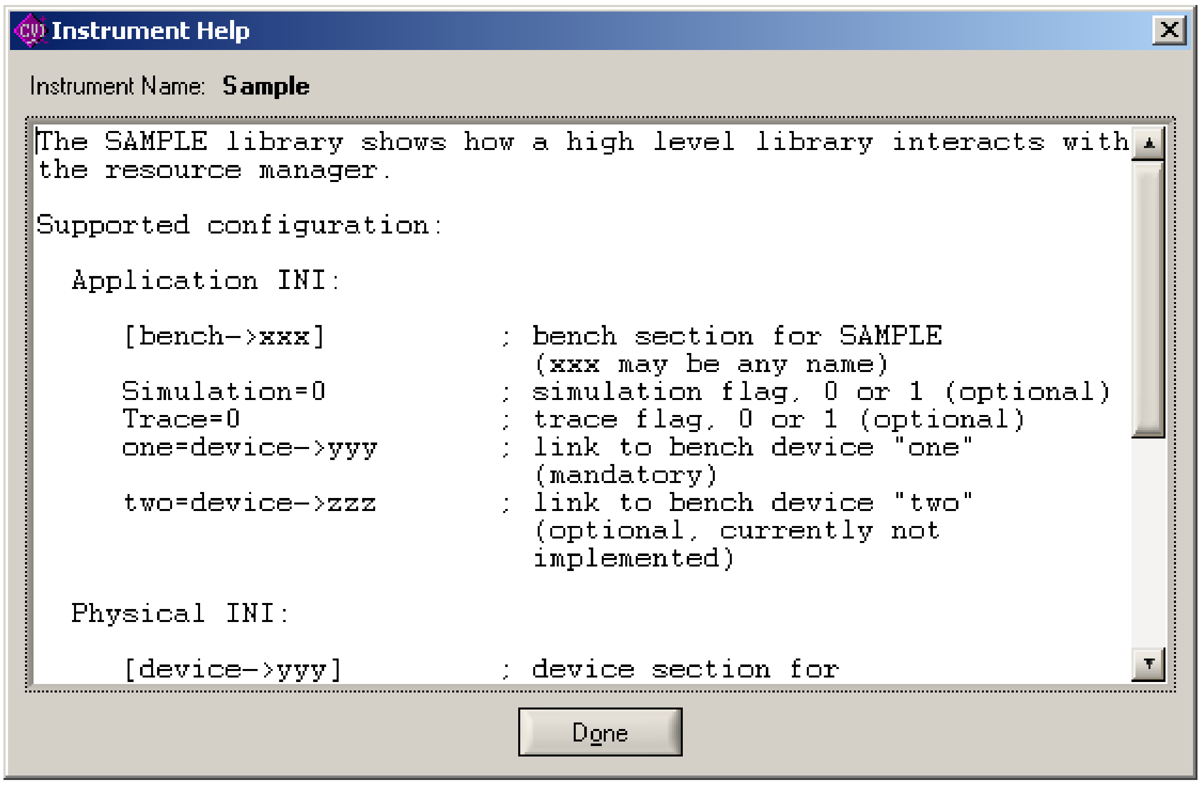


```
0
bench->GSM"
&resourceId
rorOccurred
&errorCode
errorMessag
```

```
SAMPLE_Setup (0, "bench->GSM", &resourceId, &errorOccurred,
                &errorCode, errorMessage);
```

Figure 9-4 Panel window of the SAMPLE_Setup function

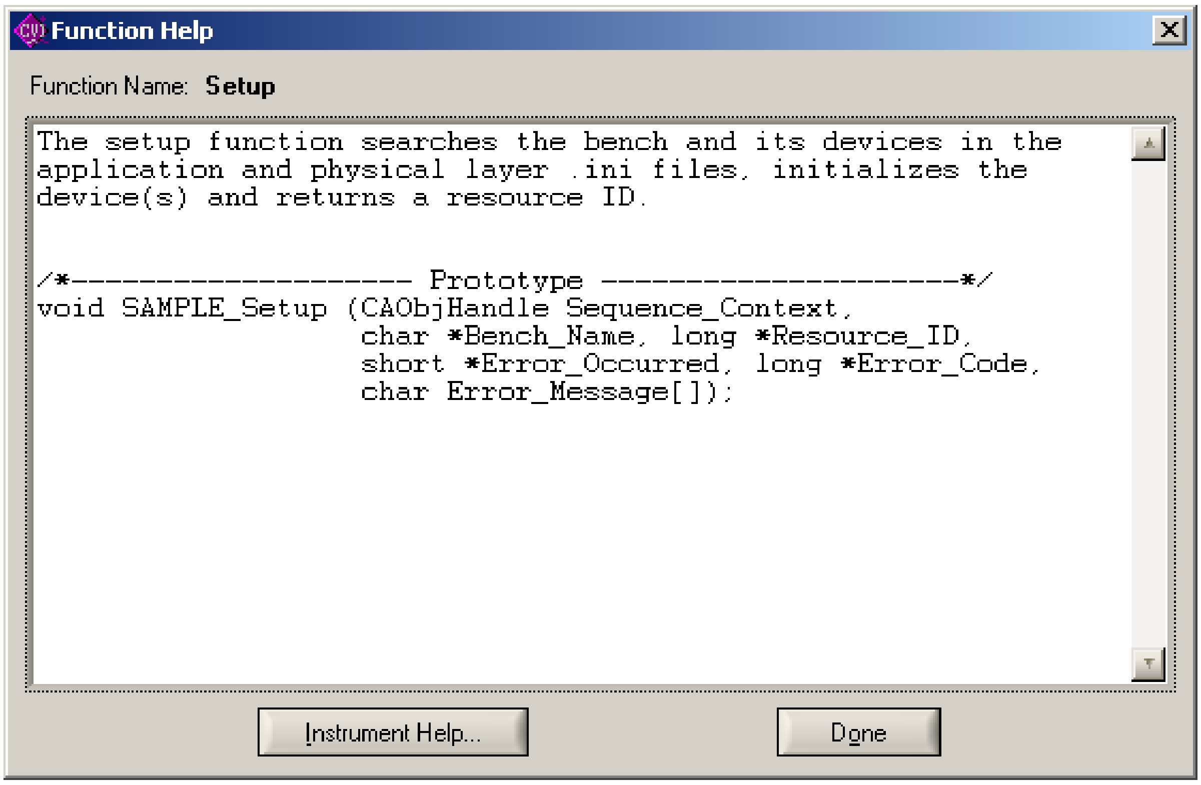
The online-help system provides a short description for the library, for each function and for each parameter:



```
The SAMPLE library shows how a high level library interacts with
the resource manager.

Supported configuration:

  Application INI:

     [bench->xxx]            ; bench section for SAMPLE
                               (xxx may be any name)
     Simulation=0            ; simulation flag, 0 or 1 (optional)
     Trace=0                 ; trace flag, 0 or 1 (optional)
     one=device->yyy         ; link to bench device "one"
                               (mandatory)
     two=device->zzz         ; link to bench device "two"
                               (optional, currently not
                               implemented)

  Physical INI:

     [device->yyy]           ; device section for
```

Figure 9-5 Library Help

Library Help (in CVI called 'Function Help') gives an overview of the purpose of the library, lists the hardware requirements (if applicable) and describes the supported entries in the application and physical layer INI files.

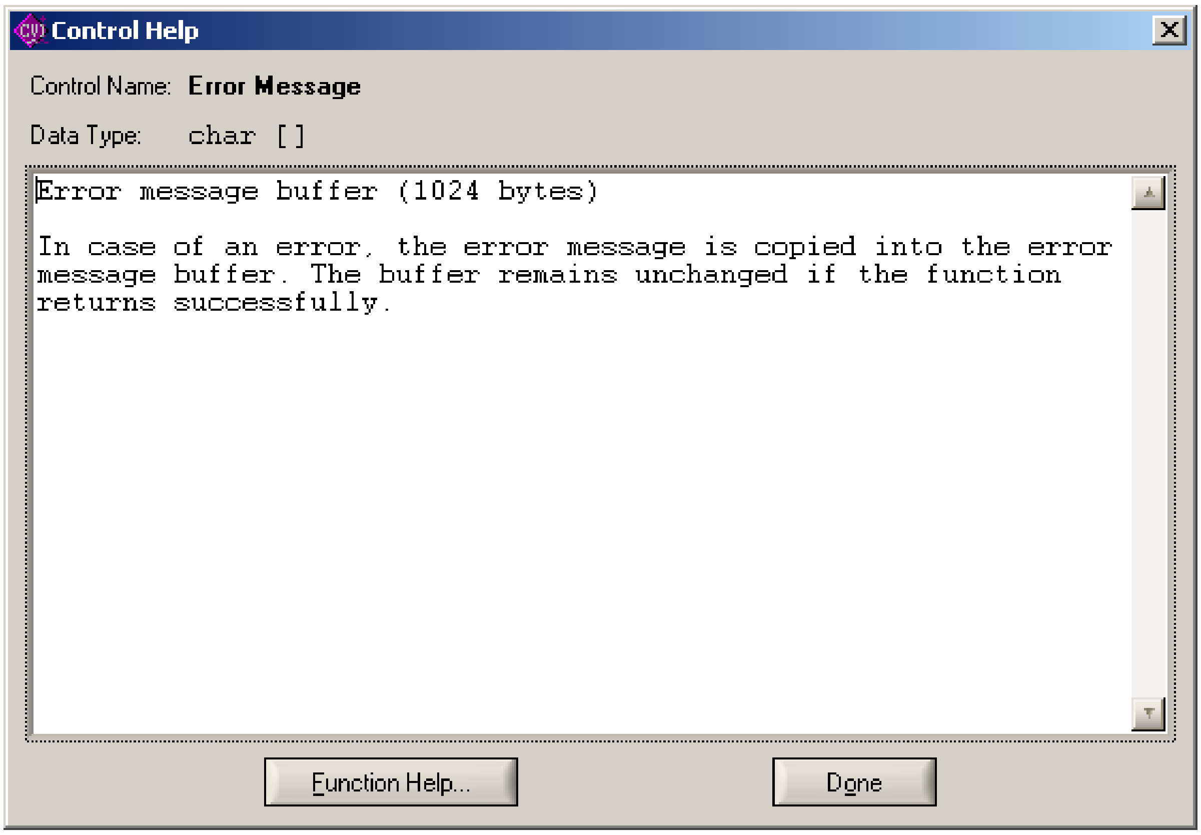


Function Name: **Setup**

```
The setup function searches the bench and its devices in the
application and physical layer .ini files, initializes the
device(s) and returns a resource ID.


/*--------------------- Prototype ---------------------*/
void SAMPLE_Setup (CAObjHandle Sequence_Context,
                   char *Bench_Name, long *Resource_ID,
                   short *Error_Occurred, long *Error_Code,
                   char Error_Message[]);
```

Figure 9-6 Function Help

Function Help gives a short summary of the task of the function.



Control Name: Error Message

Data Type: char []

```
Error message buffer (1024 bytes)

In case of an error, the error message is copied into the error
message buffer. The buffer remains unchanged if the function
returns successfully.
```

Figure 9-7 Parameter Help

9.5 Software detailed design

9.5.1 Coding rules

9.5.1.1 Language

The R&S GTSL software is developed in many different locations worldwide. This means that the language for code, comments and documentation has to be English (even for variable names, typedefs, define etc.)

9.5.1.2 Programming environment

Code is written in ANSI C. The target compiler is LabWindows/CVI version 5.5, where the compiler options are set to 'Visual C/C++ compatibility'. Non-standard extensions may be used only if absolutely necessary (like the calling convention and DLL export keywords in the function prototype).

A high-level library must meet the design specifications for a LabWindows/CVI device driver library, i.e. it is to be supplied with:

- an include file (.h file)
- a function panel (.fp file)
- a windows help file (.hlp file)
- an import library file (.lib file)
- a dynamic link library file (.dll file)

9.5.1.3 Templates

The Rohde & Schwarz templates for C modules, include files and function headers shall be used.

9.5.2 Library reference

9.5.2.1 Setup function

The function call interface of the setup function is described in section 9.4.3.2.5. The setup function has the following tasks:

1. Look up the given resource name in the application layer INI-file and allocate a resource ID for it.

2. Retrieve the configuration of the bench and device(s) from the INI files.
3. Store configuration-dependent data in a memory block in the resource manager.
4. Open the session(s) in the Resource Manager, initialize the appropriate device driver(s)and store the session handle(s) in the Resource Manager.

See the source file 'sample.c' in the SAMPLE project for details on the Setup function.

```
#define SAMPLE_BENCH_DEVICE_ONE     "one"
#define SAMPLE_BENCH_DEVICE_TWO     "two"
#define SAMPLE_TYPE_ONE             "type1"
#define SAMPLE_TYPE_TWO             "type2"

typedef struct
{
   int owner;        /* memory block owner      */
   int typeOne;      /* device type one present */
   int typeTwo;      /* device type two present */
   int simulation;   /* driver simulation       */

}  BENCH_STRUCT;

void DLLEXPORT DLLSTDCALL SAMPLE_Setup ( CAObjHandle sequenceContext,
                                         char        * pBenchName,
                                         long        * pResourceId,
                                         short       * pErrorOccurred,
                                         long        * pErrorCode,
                                         char        errorMessage[] )
{
  char         buffer [BUFFER_MEDIUM] = "";
  char         traceBuffer [BUFFER_LARGE] = "";
  int          resourceType = 0;
  int          written = FALSE;
  int          matched = FALSE;
  BOOL         sessionExists = FALSE;
  BOOL         deviceLocked = FALSE;
  BOOL         trace = FALSE;
  BENCH_STRUCT * pBench = NULL;
  ViSession    sessionHandle = 0;
  ViStatus     viStatus = 0;
```

9.5.2.1.1 Allocate the resource

In the first step, the resource name given by the parameter 'benchName' must be looked up in the configuration file and a resource ID must be allocated for it. Note that 'resourceId' is a pointer to the resource ID (see function call interface above).

```
/*-------------------------------------------------------------------/
/   Allocate the resource:
/     Check whether "pBenchName" can be found in the INI files and
/     return a resource ID for the bench. This resource ID is the
/     "ticket" for any subsequent action dealing with this bench.
/-------------------------------------------------------------------*/
RESMGR_Alloc_Resource (sequenceContext, pBenchName, pResourceId,
                      pErrorOccurred, pErrorCode, errorMessage);
```

9.5.2.1.2 Retrieve the configuration

In this step, we retrieve some bench properties, bench devices and device properties and check whether the library can handle them. Here, the trace flag and the resource type are checked.

```
/*--------------------------------------------------------------------/
/  Check for trace flag:
/    The "Trace" key in the bench section is searched and its
/    value is checked. The result is recorded in the static
/    variable trace
/--------------------------------------------------------------------*/
if ( ! * pErrorOccurred )
{
  RESMGR_Compare_Value ( sequenceContext, * pResourceId, "", RESMGR_KEY_TRACE, "1", &trace,
                         pErrorOccurred, pErrorCode, errorMessage );
}
if ( ! * pErrorOccurred )
{
  RESMGR_Set_Trace_Flag ( * pResourceId, trace );

  if ( trace )
  {
    RESMGR_Trace ( ">>SAMPLE_Setup begin" );
    RESMGR_Trace ( "Tracing for SAMPLE.DLL enabled" );
    sprintf ( traceBuffer, "Bench name %s -> Resource ID %ld", pBenchName, * pResourceId );
    RESMGR_Trace ( traceBuffer );
  }
}

/*--------------------------------------------------------------------/
/   Check the resource type:
/     The SAMPLE library requires that pBenchName refers to a bench,
/     not to a single device. It is always recommended to use a bench
/     instead of a device, because a bench makes it easy to add
/     a device in future and to work with alternative devices
/     like a CMD55 or a CMU.
/--------------------------------------------------------------------*/
if ( ! * pErrorOccurred )
{
  RESMGR_Get_Resource_Type ( sequenceContext, * pResourceId, &resourceType,
                             pErrorOccurred, pErrorCode, errorMessage );
  if ( ! * pErrorOccurred )
  {
    if ( resourceType != RESMGR_TYPE_BENCH )
    {
      * pErrorOccurred = TRUE;
      * pErrorCode = SAMPLE_ERR_NOT_A_BENCH;
      formatError ( errorMessage, *pErrorCode, * pResourceId, NULL );
    }
  }
}
```

9.5.2.1.3 Store configuration-dependent data

In this step, a memory block is created and attached to the resource ID. In this example, the memory block keeps a structure where the state of the simulation flag and the presence of two bench devices is stored.

```
/*------------------------------------------------------------------------/
/   Allocate a memory block:
/     The SAMPLE library uses a structure to store some private
/     information along with the resource ID. This information can
/     be retrieved in subsequent calls to the measurement functions.
/-----------------------------------------------------------------------*/
if ( ! * pErrorOccurred )
{
  RESMGR_Alloc_Memory ( sequenceContext, * pResourceId, sizeof ( BENCH_STRUCT ),
                        ( void ** ) ( &pBench ), pErrorOccurred, pErrorCode, errorMessage );
  if ( ! * pErrorOccurred )
  {
    /* set the memory block owner field to a unique value
       which identifies the SAMPLE library as the owner of the memory.
    */
    pBench->owner = SAMPLE_ERR_BASE;

    /* set default values */
    pBench -> typeOne = FALSE;
    pBench -> typeTwo = FALSE;
    pBench -> simulation = FALSE;
  }
}


/*------------------------------------------------------------------------/
/   Check supported bench devices:
/     Look for bench device "one" and "two" and check the "Type" key
/     in the device sections of the physical layer.
/     The presence is recorded in the memory block.
/-----------------------------------------------------------------------*/

/* 1) bench device 'one', device type 'type1'

      For device 'one', we read the type into a local buffer
      and compare it with the supported values
*/
if ( ! * pErrorOccurred )
{
  RESMGR_Get_Value ( sequenceContext, * pResourceId, SAMPLE_BENCH_DEVICE_ONE,
                     RESMGR_KEY_TYPE, buffer, sizeof ( buffer ), &written,
                     pErrorOccurred, pErrorCode, errorMessage );
  if ( ! * pErrorOccurred )
  {
    if ( written == 0 )
    {
      /* no "Type" key entry found */
      * pErrorOccurred = TRUE;
      * pErrorCode = SAMPLE_ERR_NO_TYPE;
      formatError ( errorMessage, * pErrorCode, * pResourceId, SAMPLE_BENCH_DEVICE_ONE );
    }
  }
  if ( ! * pErrorOccurred )
  {
    if ( CompareStrings ( buffer, 0, SAMPLE_TYPE_ONE, 0, 0 ) == 0 )
    {
      /* found bench device "one", type "type1" */
      pBench -> typeOne = TRUE;
```

```
        if ( trace )
        {
          RESMGR_Trace ( "Bench device 'one' of 'type1' found" );
        }
      }
      else
      {
        /* no supported type key entry found */
        * pErrorOccurred = TRUE;
        * pErrorCode = SAMPLE_ERR_NO_SUPP_TYPE;
        formatError ( errorMessage, * pErrorCode, * pResourceId, SAMPLE_BENCH_DEVICE_ONE );
      }

    }
    /* 2) bench device 'two', device type 'type2'

          For device 'two' (which is optional) we can do a quick check
          by just calling the function RESMGR_Compare_Value
    */
    if ( ! * pErrorOccurred )
    {
      RESMGR_Compare_Value ( sequenceContext, * pResourceId, SAMPLE_BENCH_DEVICE_TWO,
                             RESMGR_KEY_TYPE, SAMPLE_TYPE_TWO, &matched,
                             pErrorOccurred, pErrorCode, errorMessage );
    }
    if ( ! * pErrorOccurred )
    {
      if ( matched )
      {
        /* found bench device "two", type "type2" */
        pBench -> typeTwo = TRUE;
        if ( trace )
        {
          RESMGR_Trace ( "Bench device 'two' of 'type2' found" );
        }
      }
    }
  }
  /*---------------------------------------------------------------------/
  /   Check for simulation flag:
  /     The "Simulation" key in the bench section is searched and its
  /     value is checked. The result is recorded in the memory block.
  /     Any other library-specific information like calibration,
  /     path for calibration files etc. may be handled the same way
  /     (not shown in the example).
  /---------------------------------------------------------------------*/
  if ( ! * pErrorOccurred )
  {
    RESMGR_Compare_Value ( sequenceContext, * pResourceId, "", RESMGR_KEY_SIMULATION, "1",
                           &matched, pErrorOccurred, pErrorCode, errorMessage );
    if ( ! * pErrorOccurred )
    {
      if ( matched )
      {
        pBench -> simulation = TRUE;
        if ( trace )
        {
          RESMGR_Trace ( "Simulation is enabled" );
        }
      }
    }
  }
```

9.5.2.1.4 Initialize the device driver

The following block of code must be repeated for each device driver. First, the device is locked to ensure exclusive access during the driver initialization phase. The flag 'deviceLocked' is set to remember the lock state and ensure that the device is properly unlocked at the end of the setup procedure.

```
/*--------------------------------------------------------------------/
/   Lock the device:
/     The device must be locked to prevent another process or thread
/     from accessing it.
/-------------------------------------------------------------------*/
if ( ! * pErrorOccurred )
{
  RESMGR_Lock_Device ( sequenceContext, * pResourceId, SAMPLE_BENCH_DEVICE_ONE,
                       SAMPLE_TIMEOUT, pErrorOccurred, pErrorCode, errorMessage );
  if ( ! * pErrorOccurred )
  {
    deviceLocked = TRUE;
  }
}
```

Next, a device session is opened in the resource manager.

```
/*-------------------------------------------------------------------/
/   Open the device session(s): (Not in simulation)
/     For each device in the bench, a session must be opened.
/     The code shown handles only bench device "one". The code for
/     device "two" would be just a copy with some small modifications.
/-------------------------------------------------------------------*/
if ( ! * pErrorOccurred )
{
  if ( ! pBench -> simulation )
  {
    RESMGR_Open_Session ( sequenceContext, * pResourceId, SAMPLE_BENCH_DEVICE_ONE,
                          &sessionExists, &sessionHandle,
                          pErrorOccurred, pErrorCode, errorMessage );
  }
}
/*-------------------------------------------------------------------/
/   A device session can be shared among all threads in the same
/   process. This means, that only one thread must initialize the
/   device driver and store the session handle in the resource manager.
/   Other threads can use the same session handle to communicate with
/   the device. This information is returned in the variable
/   "sessionExists". If a session already exists, we are done with
/   the job; The session handle is returned in "sessionHandle" and
/   we can start working with it. If no session exists, we must
/   initialize the device driver and store the session handle.
/
/   Opening a session increments a "usage counter" for the device
/   in the resource manager data structure. This prevents other
/   threads to call the device driver's close function which would
/   cause the session handle to become invalid as long as we are
/   using it.
/-------------------------------------------------------------------*/
```

The device driver initialization routine is called only if the bench is not in simulation mode and if no session handle was returned by the resource manager. The device driver uses the 'ResourceDesc' device property from the physical layer INI file. The VISA session handle from the device driver is then passed to the resource manager and stored there.

```
/*-------------------------------------------------------------------/
/   Initialize the device driver:
/     If a session handle does NOT exist and we are NOT in simulation
/     mode, we must now initialize the device driver and store the
/     session handle in the resource manager.
/-------------------------------------------------------------------*/
if ( ! * pErrorOccurred )
{
  if ( ( ! sessionExists ) && ( ! pBench -> simulation ) )
  {

    /*-------------------------------------------------------------------/
    /   Read the resource descriptor from the ini file:
    /     This is a mandatory key. Without it, it is not possible to
    /     initialize the device driver
    /-------------------------------------------------------------------*/
    RESMGR_Get_Value ( sequenceContext, * pResourceId, SAMPLE_BENCH_DEVICE_ONE,
                       RESMGR_KEY_RESOURCE_DESC, buffer, sizeof ( buffer ), &written,
                       pErrorOccurred, pErrorCode, errorMessage );
    if ( ! * pErrorOccurred )
    {
      if ( written == 0 )
      {
         /* no resource descriptor found */
         * pErrorOccurred = TRUE;
         * pErrorCode = SAMPLE_ERR_NO_RESOURCE_DESC;
         formatError ( errorMessage, * pErrorCode, * pResourceId, SAMPLE_BENCH_DEVICE_ONE );
      }
    }
    /*-------------------------------------------------------------------/
    /   Initialize the device driver:
    /     Calling the init function of the device driver
    /     with the resource descriptor returns a session handle
    /-------------------------------------------------------------------*/
    if ( ! * pErrorOccurred )
    {
      if ( trace )
      {
         sprintf ( traceBuffer,
                   "Initialize device driver with ResourceDesc = %s",
                   buffer );
         RESMGR_Trace ( traceBuffer );
      }
      viStatus = DRV_init ( buffer, 1, 1, &sessionHandle );
      if ( viStatus != VI_SUCCESS )
      {
         * pErrorOccurred = TRUE;
         * pErrorCode = GTSL_ERR_DRIVER_ERROR;
         formatError ( errorMessage, * pErrorCode, * pResourceId, SAMPLE_BENCH_DEVICE_ONE );
         /* append driver specific error message */
         sprintf ( errorMessage + strlen ( errorMessage ),
                   "\nDRV_init failed with status 0x%X", ( int ) viStatus );
      }
    }
    if ( ! * pErrorOccurred )
    {
      if ( trace )
      {
         sprintf ( traceBuffer, "Session handle = %ld", sessionHandle );
         RESMGR_Trace ( traceBuffer );
      }
    }
    /*-------------------------------------------------------------------/
    /   Store the session handle
    /     The session handle is stored in the resource manager.
    /     Other threads in the same process can now open a session
    /     and re-use the handle.
```

```
        /---------------------------------------------------------------------*/
        RESMGR_Set_Session_Handle ( sequenceContext, * pResourceId,
                                    SAMPLE_BENCH_DEVICE_ONE, sessionHandle,
                                    pErrorOccurred, pErrorCode, errorMessage );
      }
    }
```

9.5.2.1.5 Cleanup and error handling

Depending on the ´deviceLocked´ flag, the device must be unlocked. If an error was detected in the setup function, the error code and message are written to the trace file.

```
  /*----------------------------------------------------------------------/
  /   Cleanup and error handling
  /---------------------------------------------------------------------*/
  /*----------------------------------------------------------------------/
  /   Unlock the device
  /     The device must be unlocked, otherwise it cannot be accessed
  /     from another thread or process.
  /---------------------------------------------------------------------*/

  if ( deviceLocked )
  {
    /*--------------------------------------------------------------------/
    /   be careful not to overwrite the error info in case of
    /   a problem before, otherwise it is not reported
    /   to the user. Local variables are therefore used here.
    /-------------------------------------------------------------------*/
    short occ = FALSE;
    long  code = 0;
    char  msg[GTSL_ERROR_BUFFER_SIZE] = "";

    RESMGR_Unlock_Device ( sequenceContext, * pResourceId, SAMPLE_BENCH_DEVICE_ONE,
                           &occ, &code, msg );
    if ( ( occ ) && ( ! * pErrorOccurred ) )
    {
      /* An error occurred during unlock. We report this error ONLY
         if no previous error exists.
      */
      * pErrorOccurred = occ;
      * pErrorCode = code;
      strcpy ( errorMessage, msg );
    }
  }
  if ( trace )
  {
    if ( * pErrorOccurred )
    {
       sprintf ( traceBuffer, "Error %ld : %s", * pErrorCode, errorMessage );
       RESMGR_Trace ( traceBuffer );
    }
    RESMGR_Trace ( "<<SAMPLE_Setup end" );
  }
}
```

9.5.2.2 Cleanup function

The function call interface of the cleanup function is described in section 9.4.3.2.6. The cleanup function has the following tasks:

- Close the device driver(s)
- Free the session in the resource manager
- Release the memory block
- Free the resource ID

See the source file 'sample.c' in the SAMPLE project for details on the Cleanup function.

```
void DLLEXPORT DLLSTDCALL SAMPLE_Cleanup ( CAObjHandle sequenceContext,
                                           long        resourceId,
                                           short       * pErrorOccurred,
                                           long        * pErrorCode,
                                           char        errorMessage[] )
{
  ViSession    sessionHandle = 0;
  ViStatus     viStatus = 0;
  BENCH_STRUCT * pBench = NULL;
  int          canClose = FALSE;
  int          deviceLocked = FALSE;
  char         traceBuffer [BUFFER_LARGE] = "";
  BOOL         trace = FALSE;

  trace = RESMGR_Get_Trace_Flag ( resourceId );
  * pErrorOccurred = FALSE;
  * pErrorCode = 0;

  if ( trace != 0 )
  {
    RESMGR_Trace ( ">>SAMPLE_Cleanup begin" );
  }
```

9.5.2.2.1 Retrieve configuration data

Configuration data is stored in a memory block during the Setup function. A pointer to this memory block is retrieved. The cleanup actions have to be taken depending on the configuration data.

```
/*-------------------------------------------------------------------/
/   Retrieve the memory pointer:
/     Get a pointer to the memory block to check the configuration
/-------------------------------------------------------------------*/
RESMGR_Get_Mem_Ptr ( sequenceContext, resourceId, ( void * * ) ( &pBench ),
                     pErrorOccurred, pErrorCode, errorMessage );
/*-------------------------------------------------------------------/
/   Check for memory block owner:
/     To be sure that the given resource ID belongs to the SAMPLE
/     library, we check the "owner" field of the memory block if it
/     contains the "magic number" we have stored there in the
/     SAMPLE_Setup function
/-------------------------------------------------------------------*/
if ( ! * pErrorOccurred )
{
  if ( pBench -> owner != SAMPLE_ERR_BASE )
  {
    * pErrorOccurred = TRUE;
    * pErrorCode = GTSL_ERR_WRONG_RESOURCE_ID;
    formatError ( errorMessage, * pErrorCode, resourceId, NULL );
  }
}
```

9.5.2.2.2 Retrieve the session handle

The session handle for each device must be retrieved before the device driver can be closed. The device must be locked to ensure exclusive access.

```
/*--------------------------------------------------------------------/
/   check if type one is present:
/    (code for typeTwo is not included in this example)
/--------------------------------------------------------------------*/
if ( ! * pErrorOccurred )
{
  if ( pBench -> typeOne )
  {
    /*--------------------------------------------------------------------/
    /   Lock the device:
    /     prevent access from other threads or processes to the device
    /--------------------------------------------------------------------*/
    RESMGR_Lock_Device ( sequenceContext, resourceId, SAMPLE_BENCH_DEVICE_ONE,
                         SAMPLE_TIMEOUT, pErrorOccurred, pErrorCode, errorMessage );
    if ( ! * pErrorOccurred )
    {
      deviceLocked = TRUE;
    }
    /*--------------------------------------------------------------------/
    /   Get the session handle:
    /     We need the session handle to close the instrument driver
    /     unless we are in simulation mode.
    /--------------------------------------------------------------------*/
    if ( ! * pErrorOccurred )
    {
      if ( ! pBench -> simulation )
      {
        RESMGR_Get_Session_Handle ( sequenceContext, resourceId,
                                    SAMPLE_BENCH_DEVICE_ONE, &sessionHandle,
                                    pErrorOccurred, pErrorCode, errorMessage );
      }
    }
```

9.5.2.2.3 Close the session and the driver

First, the session must be closed in the resource manager. If there is no other session open to this device, the device driver must be closed too.

```
/*-------------------------------------------------------------------/
/   Close the session (not in simulation)
/   - Tell the resource manager, that we no longer use this device.
/     If the usage count reachs zero (i.e. no other thread uses
/     the session handle) the resource manager tells us that we
/     are responsible for closing the instrument session. If any
/     other thread in the same process has an open session to the
/     device, it is up to HIM to call the close function of the
/     driver. In this case, it would be a failure if we closed
/     the driver, because this action invalidates the session
/     handle. The other thread would get into large trouble when
/     trying to communicate with the device next time!
/-------------------------------------------------------------------*/
if ( ! * pErrorOccurred )
{
  if ( ! pBench -> simulation )
  {
    RESMGR_Close_Session ( sequenceContext, resourceId, SAMPLE_BENCH_DEVICE_ONE,
                           &canClose, pErrorOccurred, pErrorCode, errorMessage );
  }
}
if ( ! * pErrorOccurred )
{
  if ( ( canClose ) && ( ! pBench -> simulation ) )
  {
    /*-------------------------------------------------------------------/
    /   Close the driver
    /     We are the last user of the device, so we are responsible
    /     for closing the driver
    /-------------------------------------------------------------------*/
    if ( trace )
    {
      RESMGR_Trace ( "Close the device driver" );
    }
    viStatus = DRV_close ( sessionHandle );
    if ( viStatus != VI_SUCCESS )
    {
      *pErrorOccurred = TRUE;
      *pErrorCode = GTSL_ERR_DRIVER_ERROR;
      formatError ( errorMessage, *pErrorCode, resourceId,
                    SAMPLE_BENCH_DEVICE_ONE );
      /* append driver specific error message */
      sprintf ( errorMessage + strlen(errorMessage),
                "\nDRV_close failed with status 0x%X", ( int ) viStatus );
    }
  }
}
```

9.5.2.2.4 Free the resource

The device is unlocked, the memory block is released and the resource ID is freed.

```
    /*--------------------------------------------------------------------/
    /   Unlock the device
    /--------------------------------------------------------------------*/
    if ( ! * pErrorOccurred )
    {
      RESMGR_Unlock_Device ( sequenceContext, resourceId, SAMPLE_BENCH_DEVICE_ONE,
                             pErrorOccurred, pErrorCode, errorMessage);
      if ( ! * pErrorOccurred )
      {
        deviceLocked = FALSE;
      }
    }
  }
}
/*--------------------------------------------------------------------/
/   Dispose memory:
/     Free the memory block associated with the resource ID.
/     Note that pBench is no longer valid now because it points
/     to dynamic memory that has been released!
/--------------------------------------------------------------------*/
if ( ! * pErrorOccurred )
{
  RESMGR_Free_Memory ( sequenceContext, resourceId, pErrorOccurred,
                       pErrorCode, errorMessage );
}
pBench = NULL;

/*--------------------------------------------------------------------/
/   Free resource:
/     The resource ID (our "ticket") is given back to the resource
/     manager and may be reused in a subsequent RESMGR_Alloc_Resource
/     call.
/--------------------------------------------------------------------*/
if ( ! * pErrorOccurred )
{
  RESMGR_Free_Resource ( sequenceContext, resourceId, pErrorOccurred,
                         pErrorCode, errorMessage );
}
if ( ! * pErrorOccurred )
{
  if ( trace )
  {
    sprintf ( traceBuffer, "Free Resource ID %ld", resourceId );
    RESMGR_Trace ( traceBuffer );
  }
}
```

9.5.2.2.5 Cleanup and error handling

The device must be unlocked unless this has been done before.

```
  /*-------------------------------------------------------------------/
  /   Cleanup and error handling
  /-------------------------------------------------------------------*/

  /*-------------------------------------------------------------------/
  /   Unlock the device, if there was an error.
  /     If no error occured before, then the device is already unlock
  /     (see code above).
  /-------------------------------------------------------------------*/
  if ( deviceLocked )
  {
    /*-------------------------------------------------------------------/
    /   be careful not to overwrite the error info in case of
    /   a problem before, otherwise it is not reported
    /   to the user. Local variables are therefore used here.
    /-------------------------------------------------------------------*/
    short occ = FALSE;
    long  code = 0;
    char  msg[GTSL_ERROR_BUFFER_SIZE] = "";

    RESMGR_Unlock_Device ( sequenceContext, resourceId, SAMPLE_BENCH_DEVICE_ONE,
                           &occ, &code, msg );
    if ( ( occ ) && ( ! * pErrorOccurred ) )
    {
      /* An error occurred during unlock. We report this error ONLY
         if no previous error exists.
      */
      * pErrorOccurred = occ;
      * pErrorCode = code;
      strcpy ( errorMessage, msg );
    }
  }
  if ( trace )
  {
    if ( * pErrorOccurred )
    {
       sprintf ( traceBuffer, "Error %ld : %s", * pErrorCode, errorMessage );
       RESMGR_Trace ( traceBuffer );
    }
    RESMGR_Trace ( "<<SAMPLE_Cleanup end" );
  }
}
```

9.5.2.3 Library version function

The function call interface of the setup function is described in section 9.4.3.2.7. The library version function returns a text string indicating the library name and the version number of the library. This string is copied into the string buffer 'libraryVersion'. The length must not exceed 80 characters, including the terminating null character.

The library name is the same as the library prefix.

Example: ”SAMPLE 01.02”

See chapter 9.5.4.6 for details about version number handling.

```
static const char LIB_VERSION[] = "SAMPLE 02.00"; /* Library Version String */

void DLLEXPORT DLLSTDCALL SAMPLE_Lib_Version ( CAObjHandle sequenceContext,
                                               char        libraryVersion[],
                                               short       * pErrorOccurred,
                                               long        * pErrorCode,
                                               char        errorMessage[] )
{
  * pErrorOccurred = FALSE;
  * pErrorCode = 0;
  strcpy ( libraryVersion, LIB_VERSION );
}
```

9.5.2.4 Measurement functions

The measurement functions follow the call interface described in section 9.4.3.2. The term 'measurement function' includes not only functions that take measurements, but also any function which communicates with a stimulus device, a measurement device or the UUT. Such a function may modify device settings and/or take one or more measurements, dealing with one or more devices and drivers.

The measurement function in general has the following tasks:

- Get a pointer to the memory block and retrieve configuration data
- Handle simulation mode
- Get the session handle(s) for the device(s)
- Call the device driver function(s)
- Return the measured value(s)

See the source file 'sample.c' in the SAMPLE project for details on the Measurement function.

The following example shows a simple measurement function, taking no additional parameters and returning a single measured value.

```
void DLLEXPORT DLLSTDCALL SAMPLE_MeasFunc ( CAObjHandle sequenceContext,
                                            long        resourceId,
                                            double      * measuredValue,
                                            short       * pErrorOccurred,
                                            long        * pErrorCode,
                                            char        errorMessage[] )
{
  ViSession    sessionHandle = 0;
  ViStatus     viStatus = 0;
  BENCH_STRUCT * pBench = NULL;
  int          deviceLocked = FALSE;
  int          retVal = 0;
  int          retFromFct = 0;
  char         traceBuffer [BUFFER_LARGE] = "";
  BOOL         trace = FALSE;

  * pErrorOccurred = FALSE;
  * pErrorCode = 0;

  trace = RESMGR_Get_Trace_Flag ( resourceId );

  if ( trace )
  {
    RESMGR_Trace ( ">>SAMPLE_MeasFunc begin" );

  }
```

9.5.2.4.1 Retrieve configuration data

```
/*-------------------------------------------------------------------/
/   Retrieve the memory pointer:
/     The SAMPLE library keeps important information about its
/     configuration in the memory block. This information is
/     associated with the resource ID. If SAMPLE_Setup is called
/     more than once, it returns different resource IDs. That means
/     that each resource ID (i.e. each call of SAMPLE_Setup) has its
/     own copy of the memory block.
/-------------------------------------------------------------------*/
RESMGR_Get_Mem_Ptr ( sequenceContext, resourceId, ( void * * ) ( &pBench ),
                     pErrorOccurred, pErrorCode, errorMessage );
/*-------------------------------------------------------------------/
/   Check for memory block owner:
/     To be sure that the given resource ID belongs to the SAMPLE
/     library, we check the "owner" field of the memory block if it
/     contains the "magic number" we have stored there in the
/     SAMPLE_Setup function
/-------------------------------------------------------------------*/
if ( ! * pErrorOccurred )
{
  if ( pBench -> owner != SAMPLE_ERR_BASE )
  {
    * pErrorOccurred = TRUE;
    * pErrorCode = GTSL_ERR_WRONG_RESOURCE_ID;
    formatError ( errorMessage, * pErrorCode, resourceId, NULL );
  }
}
```

9.5.2.4.2 Handle simulation mode

In simulation mode, a default measured value is returned without any interaction with the device driver.

```
/*-------------------------------------------------------------------/
/   Check for simulation:
/     In simulation mode, the device driver must not be called.
/     Just return some value.
/-------------------------------------------------------------------*/
if ( ! * pErrorOccurred )
{
  if ( pBench -> simulation )
  {
    /* simulation value */
    *measuredValue = STD_SIMU_RESULT;

    if ( trace )
    {
       sprintf ( traceBuffer, "Simulation value = %f", *measuredValue );
       RESMGR_Trace ( traceBuffer );
    }
  }
  else
  {
```

9.5.2.4.3 Retrieve session handle

Depending on the configuration information, the session handles for the required devices must be retrieved from the Resource Manager. The device must be locked.

```
/*-------------------------------------------------------------------/
/   Check for currently used device:
/     The BENCH_STRUCT keeps information about the devices that
/     have been configured during SAMPLE_Setup. We check if device
/     typeOne has been setup.
/     (Code for typeTwo is not included in this example)
/------------------------------------------------------------------*/

if ( pBench -> typeOne )
{
  /*-------------------------------------------------------------------/
  /   Lock the device:
  /     The device must be locked to prevent another process or
  /     thread from accessing it.
  /------------------------------------------------------------------*/
  RESMGR_Lock_Device ( sequenceContext, resourceId,
                       SAMPLE_BENCH_DEVICE_ONE, SAMPLE_TIMEOUT,
                       pErrorOccurred, pErrorCode, errorMessage );
  if ( ! * pErrorOccurred )
  {
    deviceLocked = TRUE;
  }
  /*-------------------------------------------------------------------/
  /   Get the session handle
  /     Retrieve the session handle for device "one".
  /------------------------------------------------------------------*/
  if ( ! * pErrorOccurred )
  {
    RESMGR_Get_Session_Handle ( sequenceContext, resourceId,
                               SAMPLE_BENCH_DEVICE_ONE, &sessionHandle,
                               pErrorOccurred, pErrorCode, errorMessage );
  }
```

9.5.2.4.4 Device driver call

The device driver is called with the session handle from the Resource Manager.

```
/*-------------------------------------------------------------------/
/   Call the driver function(s):
/     It may be necessary to call more than one function.
/     Because the device is locked, we can be sure that no other
/     thread or process can access the device now.
/------------------------------------------------------------------*/
/*-------------------------------------------------------------------/
/   Driver function call, Sample 1: (e.g. for CMD)
/------------------------------------------------------------------*/
if ( ! * pErrorOccurred )
{
  viStatus = DRV_meas_1 ( sessionHandle, measuredValue );
  if ( viStatus != VI_SUCCESS )
  {
    * pErrorOccurred = TRUE;
    * pErrorCode = GTSL_ERR_DRIVER_ERROR;
    formatError ( errorMessage, *pErrorCode, resourceId,
                  SAMPLE_BENCH_DEVICE_ONE );
    /* append driver specific error message */
    sprintf ( errorMessage + strlen(errorMessage),
              "\nDRV_meas failed with status 0x%X\n", ( int ) viStatus );
    /* read and append driver specific error message */
    retFromFct = DRV_ErrorMessage ( sessionHandle, viStatus,
                                    errorMessage + strlen(errorMessage) );
  }
  else
  {
    if ( trace )
    {
      sprintf ( traceBuffer, "Measured value = %f", *measuredValue );
      RESMGR_Trace ( traceBuffer );
    }
  }
}
/*-------------------------------------------------------------------/
/   End of Driver function call, Sample 1
/------------------------------------------------------------------*/
```

```
      /*--------------------------------------------------------------------/
      /   Driver function call, Sample 2: (e.g. for CMU)
      /--------------------------------------------------------------------*/
      if ( ! * pErrorOccurred )
      {
        viStatus = DRV_meas_2 ( sessionHandle, measuredValue, errorMessage );
        if ( viStatus != VI_SUCCESS )
        {
          * pErrorOccurred = TRUE;
          * pErrorCode = GTSL_ERR_DRIVER_ERROR;
          formatError ( errorMessage, *pErrorCode, resourceId,
                        SAMPLE_BENCH_DEVICE_ONE );
        }
        else
        {
          if ( trace )
          {
            sprintf ( traceBuffer, "Measured value = %f", *measuredValue );
            RESMGR_Trace ( traceBuffer );
          }
        }
      }
      /*--------------------------------------------------------------------/
      /   End of Driver function call, Sample 2
      /--------------------------------------------------------------------*/
    }
  }
}
```

9.5.2.4.5 Cleanup and error handling

The device must be unlocked if it has been locked before.

```
  /*-------------------------------------------------------------------/
  /   Cleanup and error handling
  /-------------------------------------------------------------------*/

  /*-------------------------------------------------------------------/
  /   Unlock the device
  /     The device must be unlocked, otherwise it cannot be accessed
  /     from another thread or process.
  /-------------------------------------------------------------------*/

  if ( deviceLocked )
  {
    /*-------------------------------------------------------------------/
    /   be careful not to overwrite the error info in case of
    /   a problem before, otherwise it is not reported
    /   to the user. Local variables are therefore used here.
    /-------------------------------------------------------------------*/
    short occ = FALSE;
    long  code = 0;
    char  msg[GTSL_ERROR_BUFFER_SIZE] = "";

    RESMGR_Unlock_Device ( sequenceContext, resourceId, SAMPLE_BENCH_DEVICE_ONE,
                           &occ, &code, msg );
    if ( ( occ ) && ( ! * pErrorOccurred ) )
    {
      /* An error occurred during unlock. We report this error ONLY
         if no previous error exists.
      */
      * pErrorOccurred = occ;
      * pErrorCode = code;
      strcpy ( errorMessage, msg );
    }
  }
  if ( trace )
  {
    if ( * pErrorOccurred )
    {
       sprintf ( traceBuffer, "Error %ld : %s", * pErrorCode, errorMessage );
       RESMGR_Trace ( traceBuffer );
    }
    RESMGR_Trace ( "<<SAMPLE_MeasFunc end" );
  }
}
```

9.5.3 Resource description

The physical and application layer INI-files contain the resource description for the test system. The general structure of these files is described in Section 5 of the document. The key names and the meaning of their values, however, are defined by the high-level library that uses them. Because most high-level libraries perform similar tasks, there is a need for standardization of the resource description.

The following example shows some library-specific entries in boldface:

```
[LogicalNames]
GSM = bench->Radiocom_GSM

[bench->Radiocom_GSM]
Description = Bench for GSM library
RadioComTester = device->CMD55
Simulation=0

[device->CMD55]
Description = Radio Communication Tester CMD55
Type = CMD55
ResourceDesc = GPIB0::15
```

There are three different types of resource entries:

1. A link to a device entry (bench device). The key RadioComTester (left side) identifies a device type, which is supported by the high-level library. The value device->CMD55 (right side) is the name of the device section, where the properties of the device can be found.
2. A property of a bench. The key Simulation identifies the simulation property of the bench, the value 0 means that simulation is switched off.
3. A property of a device. The key ResourceDesc identifies the resource descriptor, the value GPIB0::15 means that the device can be addressed via GPIB card 0, address 15.

Because device entries can be referenced by several high-level libraries, there must be a set of standard properties, which can be understood by each of these libraries. The same applies to bench properties like simulation or tracing, which are standard properties and are supported by each library. Only the bench device names are library-specific.

The following tables describe the standard keys, values and usage.

Key name	Remarks
Description	bench description, comment
Simulation	if set to '1', the complete bench is simulated by the library
Trace	if set to '1', tracing is enabled for the library

Table 9-1 Standard bench properties

Key name	Remarks
Description	Device description, comment
Type	Device type like CMD55, etc. (mandatory).
ResourceDesc	VISA resource descriptor like 'GPIB0::15' or 'PXI0::16::0' (mandatory). The given value must be passed to the device drivers init function (IVI and VISA).
DriverOption	Special setup string for IVI driver, e.g. for device-level simulation. If this entry is present and the device driver is an IVI driver, it must be initialized using the InitWithOptions function, passing the given setup string. If this entry is missing, the normal Init function can be used.

Table 9-2 Standard device properties

The 'Type' and 'ResourceDesc' keys are mandatory, they must be defined for each device section. All other keys are optional. Binary switches like 'Simulation' and 'Trace' are activated by the value '1' and deactivated by any other value. The switch is also deactivated if the key is not present.

The Resource Manager exports the key names in the resmgr.h file as RESMGR_KEY_... constants, e.g. RESMGR_KEY_SIMULATION for 'Simulation'. There is also a list of common device types; these string constants begin with RESMGR_DEVTYPE_...

9.5.4 Miscellaneous

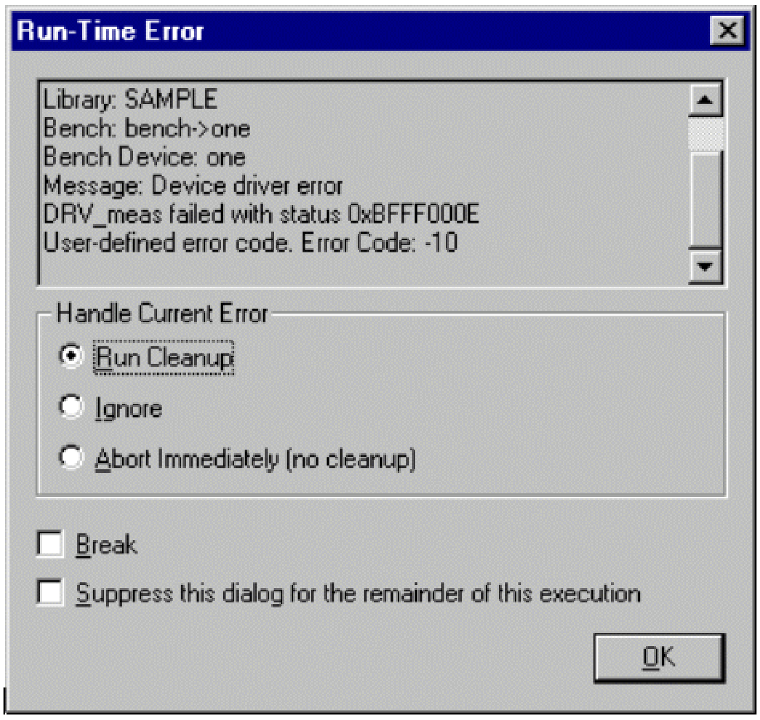
9.5.4.1 Error handling

Error handling is done through the three output parameters pErrorOccurred, pErrorCode and errorMessage of each export function of the library as described in section 9.4.3.2.3.

There are several possible sources of an error:

- Error from the high level library
- Error from a support library, e.g. the resource manager
- Error from a device driver

The pErrorCode and errorMessage should reflect the source and the reason of an error, so the user can take appropriate measures to eliminate the problem. The error codes should be unique throughout the R&S GTSL software to avoid confusion. This means, that each library has its own set of error codes. The error message should also state very clearly, which library issued the error, the name of the bench and the bench device. The following figure shows an example of an error message as it is shown in TestStand:



```
Library: SAMPLE
Bench: bench->one
Bench Device: one
Message: Device driver error
DRV_meas failed with status 0xBFFF000E
User-defined error code. Error Code: -10
```

Handle Current Error
[✓ Run Cleanup]
Ignore
Abort Immediately (no cleanup)

☐ Break
☐ Suppress this dialog for the remainder of this execution

Figure 9-8 Run-Time error message in TestStand

The first lines (not shown above) list the name of the sequence and the step. After that, the contents of the errorMessage variable is shown, followed by the decimal representation of the pErrorCode (the last line).

The error message is broken into several lines, each beginning with a prefix like 'Library', 'Bench' etc. The error message may contain additional information like the driver status code in the example above.

9.5.4.1.1 Include files

Error codes are defined in the include file of the high level library. All error codes are based on a base number, which is defined in `GTSLERR.H`. It is important to include this file in the include file for the high level library. The following code is taken from the `sample.h` file:

```
#include 'gtslerr.h'   /* GTSL error handling */

/* DEFINES ******************************************************************/

/* Error codes */
#define SAMPLE_ERR_BASE                GTSL_ERROR_BASE_SAMPLE
#define SAMPLE_ERR_NOT_A_BENCH     (SAMPLE_ERR_BASE - 1)    /* -4001 */
#define SAMPLE_ERR_NO_RESOURCE_DESC (SAMPLE_ERR_BASE - 2)    /* -4002 */
```

GTSL_ERROR_BASE_SAMPLE is defined in `gtslerr.h` as -4000. If a new high-level library is developed, a new GTSL_ERROR_BASE_XXX constant has to be defined. Use the constant GTSL_ERROR_BASE_USER as a base for error codes in your own projects. Codes between 0 and GTSL_ERROR_BASE_USER are reserved for libraries developed by Rohde & Schwarz..

The `gtslerr.h` include file defines several general-purpose error codes and messages which can be used in all high-level libraries. Library-specific error codes are defined as constants in the include-file of the library. The names start with XXX_ERR_... where XXX is the library name. The integer error numbers should be written in comment after each definition (remember not to use the '//' comment delimiter !!!). This makes it easy for the user to find the error definition by just doing a 'grep' or file search over all include files.

9.5.4.1.2 Error table

A table of error codes and corresponding error messages is kept in a static structure array in the high-level library. The type definitions GTSL_ERROR_TABLE and GTSL_ERROR_ENTRY can be found in `gtslerr.h`. This error table contains the library-specific codes and messages as well as all general-purpose codes:

```
/* Error code to message reference table */
static GTSL_ERROR_TABLE errorTable =
{
    /* library specific error codes and messages */
  {
    SAMPLE_ERR_NOT_A_BENCH,      "The given resource is not a bench" },
  {
    SAMPLE_ERR_NO_RESOURCE_DESC, "ResourceDesc entry missing in physical INI file" },
  {
    SAMPLE_ERR_NO_TYPE,          "Type entry missing in physical INI file" },
  {
    SAMPLE_ERR_NO_SUPP_TYPE,     "Type entry in physical INI file is not supported" },

    /* include common GTSL error codes and messages */
   GTSL_ERROR_CODES_AND_MESSAGES,

    /* this must be the last entry ! */
  {
    0, NULL }
};
```

The table initialization consists of three parts

- library-specific codes and corresponding messages
- common R&S GTSL error codes and messages (represented by GTSL_ERROR_CODES_AND_MESSAGES macro)
- the terminating entry { 0, NULL }

9.5.4.1.3 Signaling an error

The following examples show different cases of how errors are signaled and handled. The simplest case is an error coming from a lower level library like the resource manager:

```
  RESMGR_Get_Resource_Type ( sequenceContext, * pResourceId, &resourceType,
                             pErrorOccurred, pErrorCode, errorMessage );
    if ( ! * pErrorOccurred )
    {
      ....
    }
```

The three error parameters pErrorOccurred, pErrorCode and errorMessage are just passed from the resource manager library. In case of an error in the resource manager, all error information has already been set. The high-level library just checks the pErrorOccurred flag and skips the following code if the flag is set.

When the high-level library detects an error, it sets the pErrorOccurred flag and the pErrorCode variable and calls an internal function 'formatError' which builds the error message from the given information and copies it into errorMessage.

```
if ( resourceType != RESMGR_TYPE_BENCH )
{
  * pErrorOccurred = TRUE;
  * pErrorCode = SAMPLE_ERR_NOT_A_BENCH;
  formatError ( errorMessage, *pErrorCode, * pResourceId, NULL );
}
```

The parameter interface and the implementation of the format_error function in the SAMPLE library are just a proposal. Some libraries need some more elaborate error message generation and, perhaps, additional parameters to this function. See the following section for details on this function.

When an error occurs in a device driver function, the driver error status cannot just be returned as pErrorCode, because drivers use a somewhat different numbering scheme. Therefore, a general GTSL_ERR_DRIVER_ERROR code is returned and the error status from the driver is appended to the error message in hex display. Most drivers offer a function to convert the status value to an error string. This function is called here to append the driver-specific error message at the end:

```
  if ( ! * pErrorOccurred )
  {
    viStatus = DRV_meas_1 ( sessionHandle, measuredValue );
    if ( viStatus != VI_SUCCESS )
    {
      * pErrorOccurred = TRUE;
      * pErrorCode = GTSL_ERR_DRIVER_ERROR;
      formatError ( errorMessage, *pErrorCode, resourceId,
                    SAMPLE_BENCH_DEVICE_ONE );
      /* append driver specific error message */
      sprintf ( errorMessage + strlen(errorMessage),
                "\nDRV_meas failed with status 0x%X\n", ( int ) viStatus );
      /* read and append driver specific error message */
      retFromFct = DRV_ErrorMessage ( sessionHandle, viStatus,
                                      errorMessage + strlen(errorMessage) );
    }
  }
```

The code is very similar to the example above, except that the sprintf and DRV_error_message function calls have been added to append the driver function name, the status and the driver-specific error message.

9.5.4.1.4 Formatting the error message

The format_error function shown here is just an example of how it can be done. Each high-level library may require a different set of parameters. The main task of this function, however, is always the same: Generate the error message and put it into the error buffer.

```
static void formatError ( char buffer[],
                          int  code,
                          long resId,
                          char * benchDevice )
{
  char             * pMsg = NULL;
  char             resourceName[RESMGR_MAX_NAME_LENGTH + 1] = "";
  char             tempMsg[GTSL_ERROR_BUFFER_SIZE] = "";
  short            tempOcc = FALSE;
  long             tempCode = 0;
  int              written = 0;
  GTSL_ERROR_ENTRY * pErr = errorTable; /* pointer into error entry table */
```

First, the error message corresponding to the given code must be searched in the error table:

```
/* find the error message for a given error code */
  while ( pErr -> string != NULL )
  {
    if ( pErr -> value == code )
    {
      pMsg = pErr -> string;
      break;
    }
    pErr ++;
  }
  if ( pMsg == NULL )
  {
    /* should never happen */
    pMsg = "(no message available for this code)";
  }
```

Next, the error message is built line by line:

```
/* setup the error message */

  /* 1) Library name */
  strcpy ( buffer, GTSL_ERRMSG_PREFIX_LIBRARY );
  strcat ( buffer, GTSL_LIBRARY_NAME );
  strcat ( buffer, "\n" );
```

The name of the bench is returned by the resource manager function RESMGR_Get_Resource_Name. It takes the resource ID as a parameter. In this example, the parameter resid may be set to RESMGR_INVALID_ID. In this case, the line 'Bench: ' will not be output:

```
/* 2) Bench name, only if a valid ID is given */
  if ( resId != RESMGR_INVALID_ID )
  {
    /* read the resource name into a local buffer */
    RESMGR_Get_Resource_Name ( 0, resId, resourceName, sizeof ( resourceName ),
                               &written, &tempOcc, &tempCode, tempMsg );
    if ( ( ! tempOcc ) && ( written > 0 ) )
    {
      /* append the name */
      strcat ( buffer, GTSL_ERRMSG_PREFIX_BENCH );
      strcat ( buffer, resourceName );
      strcat ( buffer, "\n" );
    }
  }
```

If the parameter bench_device is not NULL, it is output in the next line:

```
/* 3) Bench device, if given */
  if ( benchDevice != NULL )
  {
    strcat ( buffer, GTSL_ERRMSG_PREFIX_BENCH_DEVICE );
    strcat ( buffer, benchDevice );
    strcat ( buffer, "\n" );
  }
```

Finally, the error message is appended to the message buffer:

```
/* 4) Error message */
  strcat ( buffer, GTSL_ERRMSG_PREFIX_ERRMSG );
  strcat ( buffer, pMsg );
```

9.5.4.2 Locking

The implementation of device locking is mandatory for each high-level library. Locking is done using the RESMGR_Lock_Device and RESMGR_Unlock_Device functions of the Resource Manager Library.

The Setup, Cleanup and Measurement functions must lock a device prior to the first driver call and unlock the device after the last driver call inside the function to ensure exclusive access to the device.

See RESMGR.HLP and the source code of the SAMPLE project for details.

Special care must be taken to guarantee the same number of lock and unlock calls inside the functions, also in case of an error returned by a fubction call. If a device is not unlocked correctly, it may stay locked forever, blocking another parallel test process.

The lock function requires a timeout value, i.e. the maximum amount of time which is spent waiting for the device to become free. This value must be selected depending on the maximum time it takes for a measurement with this device. A standard value is 5000 ms. The timeout value should not exceed about 20 or 30 seconds, this is the maximum time a user is willing to wait for a reaction of the system.

9.5.4.3 Tracing

During the development phase of a library module, the tracing of information is an important feature. The Resource Manager offers convenient functions for tracing, which can be used with the following benefits:

- No need to write additional code
- Tracing can be switched on and off dynamically
- Tracing from different libraries is directed to a single output file or to screen

See the description of the RESMGR_Trace function and Set/Get TraceFlag in RESMGR.HLP for details.

- Tracing may be enabled in two ways:
- by a compiler switch in the high-level library
- by the 'Trace = 1' entry in the application INI file

Using a compiler switch allows tracing during the development and debug phase. Tracing is then switched off for the release version. There is no performance loss in the release version, but there is also no easy way to re-enable tracing at a customer site in case of a problem. The library must be rebuilt with the compiler switch, which is a problem because the customer normally does not have the source code.

Using an entry in the application INI file is more convenient, because tracing may be enabled easily by adding the line 'Trace = 1' in the appropriate bench section of the INI file. The performance degradation can be kept to a minimum if a tracing flag is used in each function as shown in the following code example:

```
BOOL trace = FALSE;
```

In the Setup function, the 'Trace' property of the bench is checked and the flag is set:

```
RESMGR_Compare_Value ( sequenceContext, * pResourceId, "", RESMGR_KEY_TRACE, "1", &trace,
                       pErrorOccurred, pErrorCode, errorMessage );
```

At the beginning of each other function the actual value of the trace flag is read:

```
trace = RESMGR_Get_Trace_Flag ( resourceId );
```

Depending on the flag, tracing is done:

```
if ( trace )
{
  RESMGR_Trace ( "Close the device driver" );
}
```

Formatted output cannot be done in the RESMGR_Trace function directly. A temporary trace buffer is used to format the message first:

```
#define BUFFER_LARGE 1088
char traceBuffer [BUFFER_LARGE] = "";
if ( trace )
{
  sprintf ( traceBuffer, "Measured value = %f", *measuredValue );
  RESMGR_Trace ( traceBuffer );
}
```

9.5.4.4 Simulation

The implementation of simulation is mandatory for each high-level library. The reasons for running a library in simulation mode are:

- A sequence can be programmed and tested without hardware.
- Parallel tests can be programmed and run if only a single set of hardware is available. One test process uses the real hardware, the others run in simulation mode.
- Presentation of a test sequence to a customer without hardware (e.g. on a laptop).

Simulation is done at a very high level in the library. During simulation mode, the high-level library must not call any device driver function or any other function requiring more than the standard PC hardware resources. The library functions should return some 'typical' measured values to generate a 'Pass' condition in the calling TestStand sequence.

Simulation is enabled by the 'Simulation' keyword in the appropriate bench section. The simulation flag for each bench must be kept in the memory block associated with the resource ID. In contrast to the tracing capability (cf. section 9.5.4.1), the usage of a static simulation flag is not allowed. The following code example shows how simulation is handled.

In the Setup function, the presence of the 'Simulation' keyword is checked and the value of the simulation flag is stored in the memory block:

```
/* NOTE: error handling is omitted in this short example */

typedef struct
{
  /* ... other entries ... */
  int simulation;   /* driver simulation */
}  BENCH_STRUCT;

/* pointer to the memory block */
BENCH_STRUCT * pBench = NULL;

/* allocate the memory block */
RESMGR_Alloc_Memory ( sequenceContext, * pResourceId, sizeof ( BENCH_STRUCT ),
                     ( void * * ) ( &pBench ), pErrorOccurred, pErrorCode, errorMessage );

/* set the simulation flag in the memory block */
/* according to the 'simulation' bench property */
pBench -> simulation = FALSE;
RESMGR_Compare_Value ( sequenceContext, * pResourceId, "", RESMGR_KEY_SIMULATION, "1",
                       &matched, pErrorOccurred, pErrorCode, errorMessage );
if ( matched )
{
  pBench -> simulation = TRUE;
}
```

Each measurement function must read the value of the simulation flag and determine whether to simulate the measurement or take the measurement with the real hardware:

```
/* NOTE: error handling is omitted in this short example */

/* pointer to the memory block */
BENCH_STRUCT * pBench = NULL;

/* get the memory block pointer */
RESMGR_Get_Mem_Ptr ( sequenceContext, resourceId, ( void * * ) ( &pBench ),
                     pErrorOccurred, pErrorCode, errorMessage );

/* check for simulation flag */
if ( pBench -> simulation )
{
  *measuredValue = 1.0; /* simulation value */
}
else
{
  /* call the device driver to get a value from the instrument ... */
}
```

9.5.4.5 Bench versus device

When a high-level library requires the concurrent use of more than one device, these devices must be entered in a bench section. On the other hand, there may be a case where a library uses only a single device. Is there a reason to have a bench with only one device entry or could I just pass the name of this device to the library?

Even if only a single device is used, a bench has many advantages:

- A bench can have bench properties like simulation and tracing, a device does not have these properties.
- The library may be extended in the future to support more than one device. Migration from a device section to a bench section is harder than just adding the second device to the bench section which already exists.

Summary: It is most advantageous to work with benches, even if there is only a single device in the bench.

The high-level library must check in the Setup function to see if the resource name refers to a bench or to a device and return an error if the type is not correct:

```
RESMGR_Get_Resource_Type ( sequenceContext, * pResourceId, &resourceType,
                           pErrorOccurred, pErrorCode, errorMessage );
if ( ! * pErrorOccurred )
{
  if ( resourceType != RESMGR_TYPE_BENCH )
  {
    * pErrorOccurred = TRUE;
    * pErrorCode = SAMPLE_ERR_NOT_A_BENCH;
    formatError ( errorMessage, *pErrorCode, * pResourceId, NULL );
  }
}
```

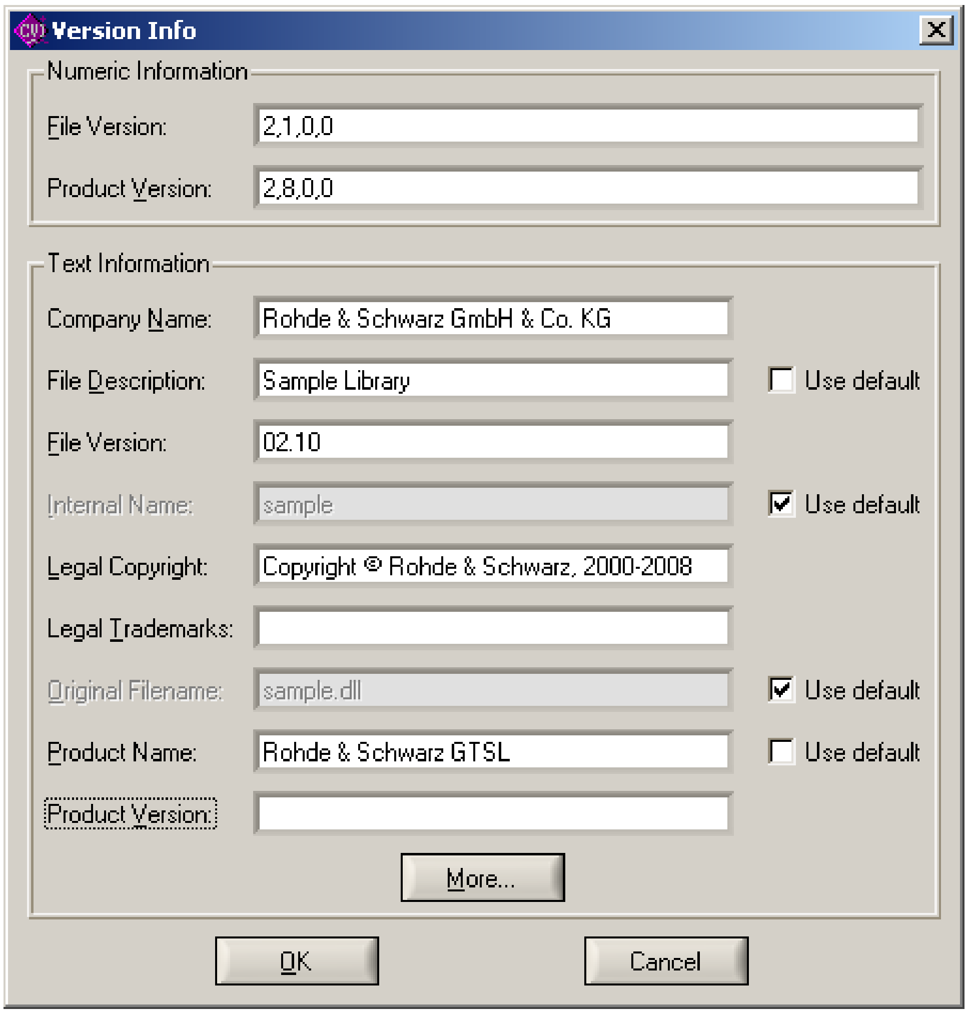
9.5.4.6 Version handling

The version number of a library is handled in two separate places. First, there is a version string which is returned by the Lib_Version Function. Second, there is a built-in version number in each DLL, which can be set during the DLL build.

The version number consists of four digits, the first two digits being separated by a decimal point. The first two digits indicate the major version of the software (with leading zero). It is incremented if the functionality changes significantly. The third and fourth digit indicate the minor version. The last digit is normally zero for a software release with new functions and is incremented for bug fix releases.

Whenever the version of a library changes, the correct version number must be specified during the build of the DLL. The version number of the DLL is very important for the software setup program. The setup program can ensure that a newer DLL version is not overwritten by an old one using the built-in version number. The version information for a DLL can be shown in the 'Properties' dialog box in the Windows 2000/XP Explorer (only the text information). Setup programs uses the numeric 'File Version' information.

The following figure shows the dialog box for the DLL version information:



File Version:	2,1,0,0	
Product Version:	2,8,0,0	
Company Name:	Rohde & Schwarz GmbH & Co. KG	
File Description:	Sample Library	☐ Use default
File Version:	02.10	
Internal Name:	sample	☑ Use default
Legal Copyright:	Copyright © Rohde & Schwarz, 2000-2008	
Legal Trademarks:		
Original Filename:	sample.dll	☑ Use default
Product Name:	Rohde & Schwarz GTSL	☐ Use default
Product Version:		

Figure 9-9 Version Info during DLL build

The following fields must be modified to build a new DLL version:

File Version (numeric)	only the first three digits are used for the version number

Product Version (numeric)	like File Version
File Version (text)	Version number like in the Library Version function
Product Version (text)	same as file version

The following example shows the hypothetical life cycle of the 'SAMPLE' library. Note that the version number must grow from release to release.

Version String	Version number	Comment
SAMPLE 01.00	1,0,0,0	first official release
SAMPLE 01.01	1,0,1,0	bug fixes
SAMPLE 01.02	1,0,2,0	more bug fixes
SAMPLE 01.10	1,1,0,0	official release
SAMPLE 01.11	1,1,1,0	bug fixes
SAMPLE 02.00	2,0,0,0	official release with new functionality
SAMPLE 02.10	2,1,0,0	official release

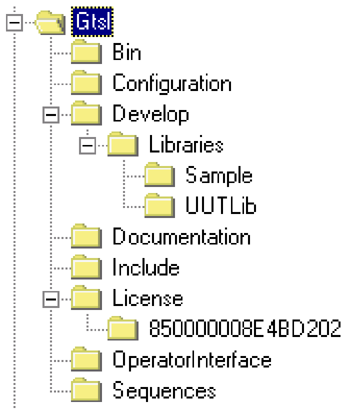
9.5.5 CVI project structure

NOTE:

Do not modify the original Library project. Instead, make a copy of the Libraries directory and make your modifications in the copy.
Be careful when you put your private DLLs in the `gtsl\bin` directory. Always keep a copy, because they may be overwritten by an update to R&S GTSL if there is a DLL with the same name. It is safer to keep your projects and DLLs in a completely separate location beside the R&S GTSL tree. Be sure to add the directory where your DLLs reside to the PATH environment variable of your computer.

9.5.5.1 Directory structure

The directory structure of the R&S GTSL software is as follows:



```
Gtsl
├─ Bin
├─ Configuration
├─ Develop
│  └─ Libraries
│     ├─ Sample
│     └─ UUTLib
├─ Documentation
├─ Include
├─ License
│  └─ 850000008E4BD202
├─ OperatorInterface
└─ Sequences
```

Figure 9-10 R&S GTSL directory tree

GTSL is the root directory of the tree. It may be located anywhere in the system, e.g. under `C:\Program Files\GTSL` or `E:\GTSL`.

Bin contains the following files for all libraries and drivers:

- .DLL, Dynamic Link Library
- .LIB, Import Library for the DLL, Microsoft C/C++ compatible
- .FP and SUB, CVI Function Panel
- .HLP, Windows Help File
- .CDD, CVI DLL Debugger Information (for development versions only)

Include contains the include (.H) files for all libraries and drivers.

Develop contains all files required to build the libraries and drivers. There is a separate subtree for the libraries and one for the drivers. Each library or driver project has its own directory, like UUTLIB and SAMPLE in Figure 9-10.

9.5.5.2 CVI project files

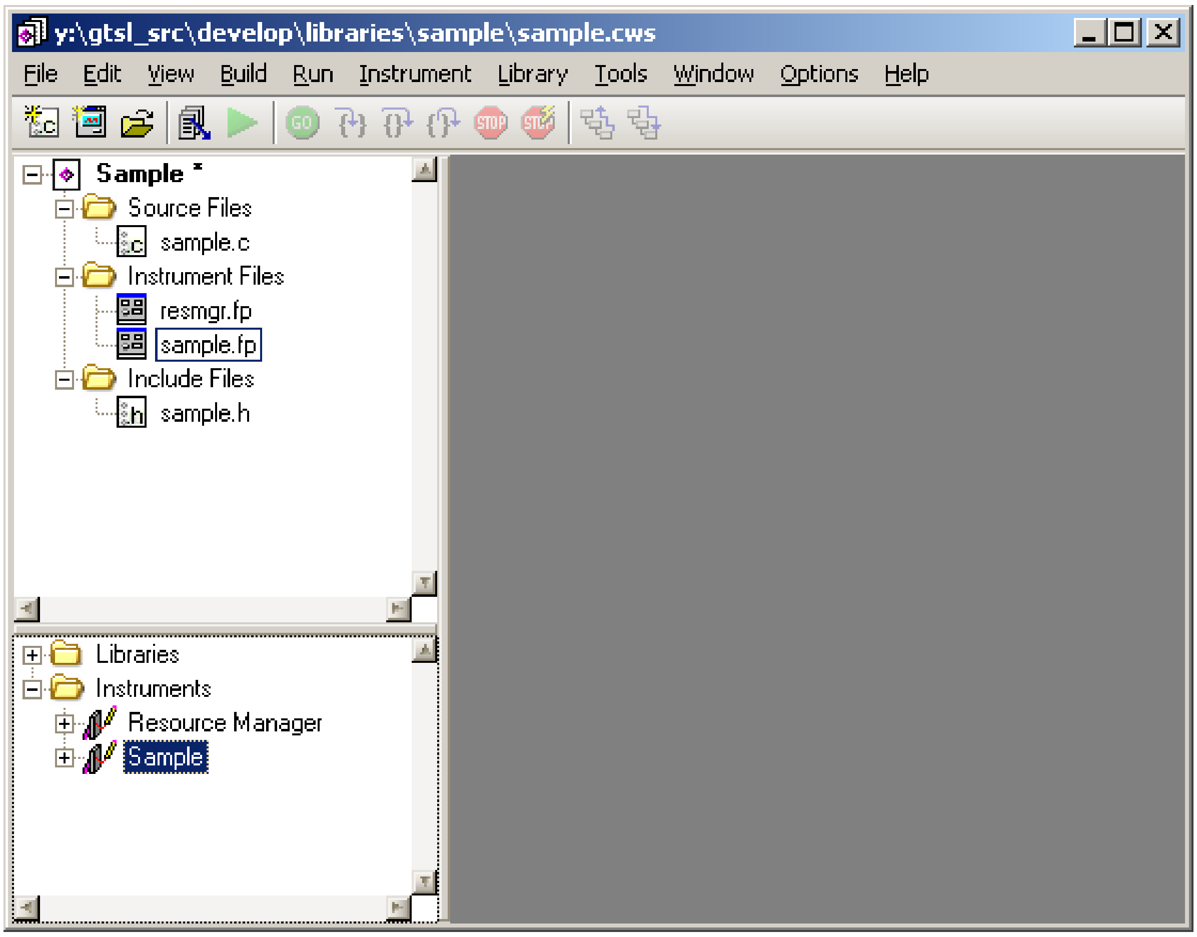


Figure 9-11 CVI project window

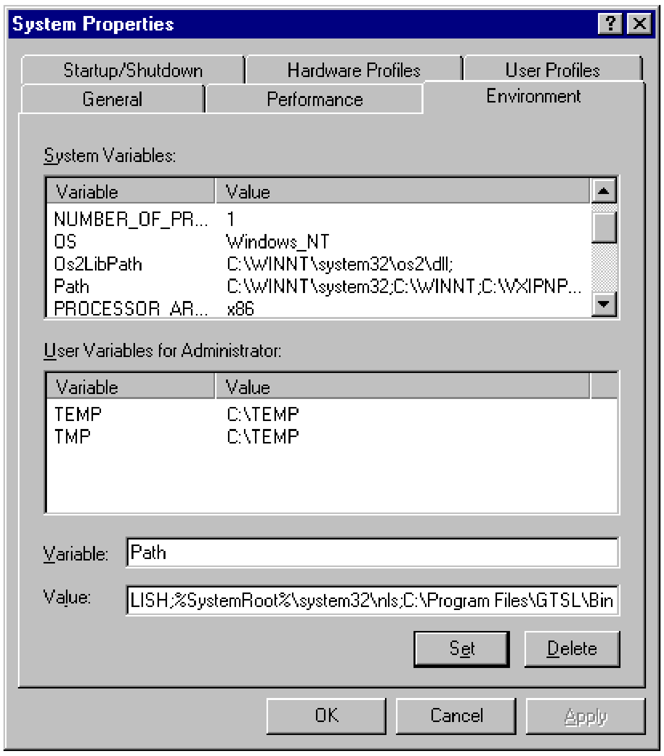
A library project under CVI contains the following files:

- The source file(s) for the library (sample.c)
- The function panel for the library (sample.fp)
- The include file containing the library export functions (sample.h)
- The function panel files for all subsidiary libraries and drivers (resmgr.fp). These files are taken from the Bin directory.

9.5.5.3 Configuration

The system must known the location of the Bin and Include directories in order to work correctly with the R&S GTSL software.

The Bin directory must be added to the PATH environment variable. This is done by the Setup program when you install R&S GTSL on your computer. When a library DLL requires a subsidiary DLL (like SAMPLE requires the RESMGR DLL), the operating system finds the subsidiary DLL using the standard DLL search algorithm. If the Bin directory is not included in the PATH, SAMPLE cannot find the RESMGR.DLL and fails if it is loaded. This must be done for a development system as well as for a run-time system.


System Properties

System Variables:

Variable	Value
NUMBER_OF_PR...	1
OS	Windows_NT
Os2LibPath	C:\WINNT\system32\os2\dll;
Path	C:\WINNT\system32;C:\WINNT;C:\VXIPNP...
PROCESSOR_AR...	x86

User Variables for Administrator:

Variable	Value
TEMP	C:\TEMP
TMP	C:\TEMP

Variable: Path

Value: LISH;%SystemRoot%\system32\nls;C:\Program Files\GTSL\Bin

Figure 9-12 Setting the PATH variable

You may add the Bin directory to the Path system variables in the upper part of the dialog box (for all users of the computer: preferred setting) or in the lower part (only for the current user).

The Include directory must be added to the list of include paths in the CVI development environment. This can be done using the Options... Include Paths command:

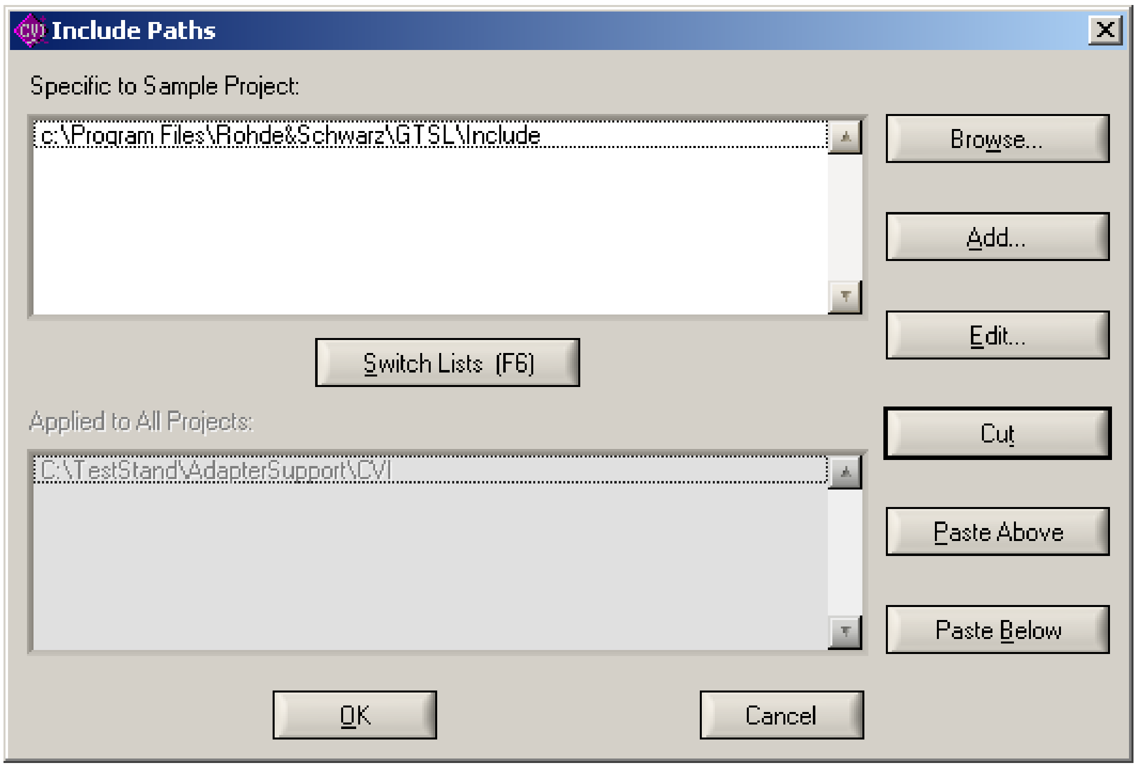


Figure 9-13 Adding the Include Path

You may enter the Include directory in the upper part of the dialog box (specific to the current project: preferred setting) or in the lower part (applied to all projects).

9.5.5.4 Building the DLL

The project settings for a high-level library are shown in the following figure:

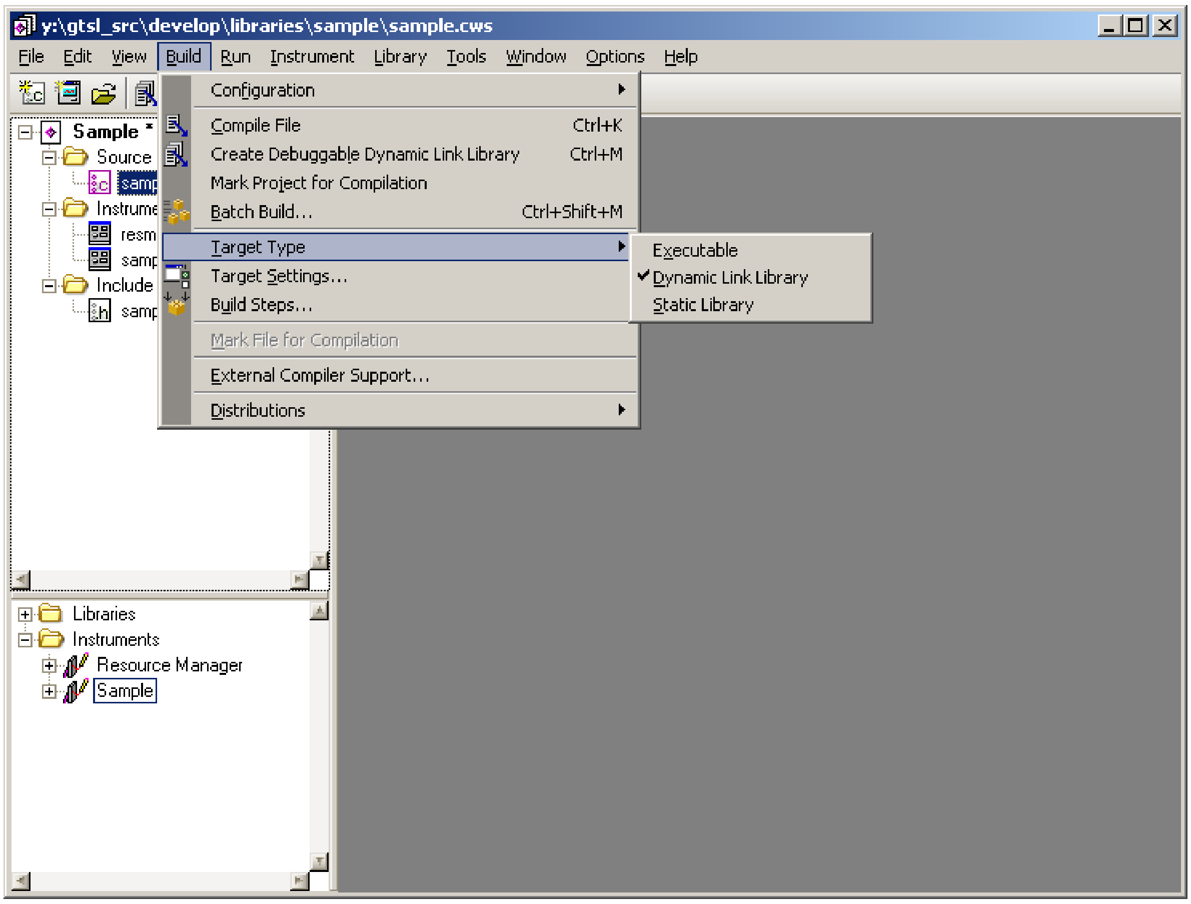
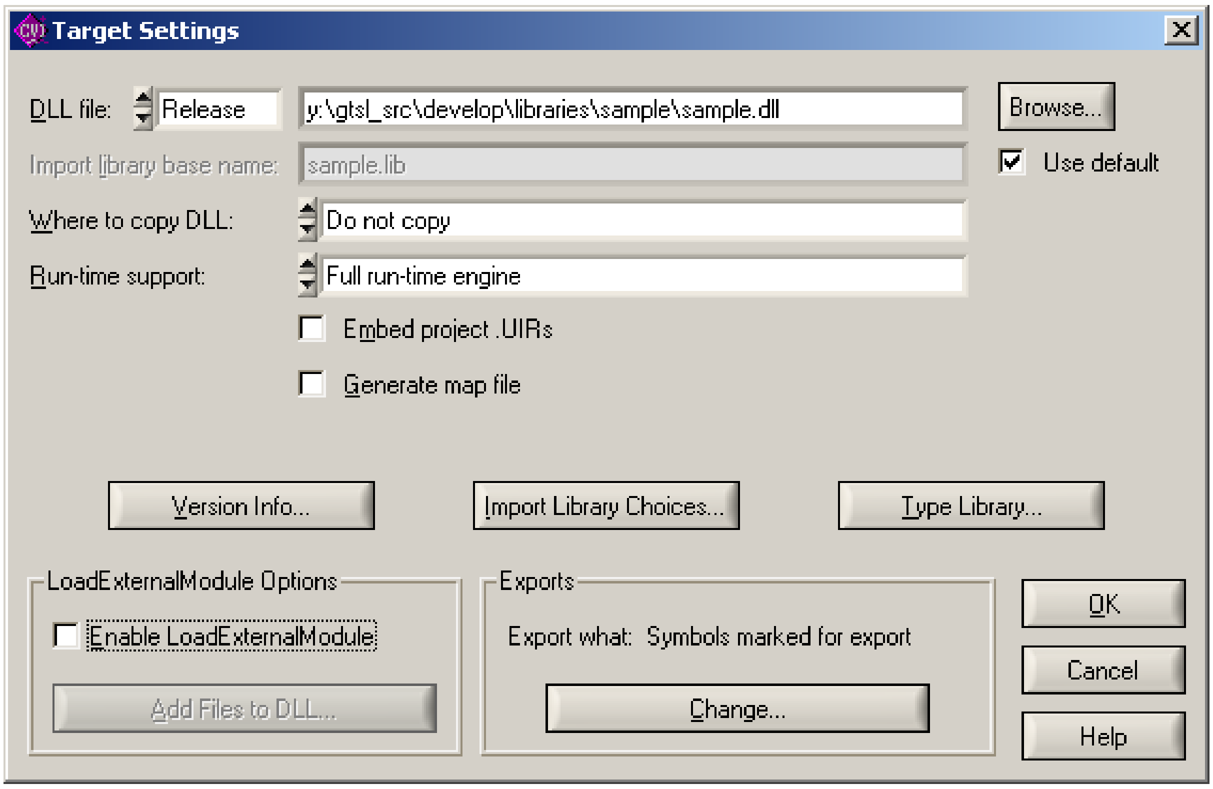

y:\gtsl_src\develop\libraries\sample\sample.cws
File Edit View Build Run Instrument Library Tools Window Options Help
Configuration
Compile File Ctrl+K
Create Debuggable Dynamic Link Library Ctrl+M
Mark Project for Compilation
Batch Build... Ctrl+Shift+M
Target Type
Target Settings...
Build Steps...
Mark File for Compilation
External Compiler Support...
Distributions
Executable
✓ Dynamic Link Library
Static Library
Sample
Source
Instrume
Include
Libraries
Instruments
Resource Manager
Sample

Figure 9-14 Build settings

The Target Type must be set to Dynamic Link Library.

Configuration is normally set to Debug during the development phase of the library, it is set to Release for the release version.

The Target Settings dialog box is shown below:


Target Settings

DLL file: Release y:\gtsl_src\develop\libraries\sample\sample.dll

Import library base name: sample.lib

Use default [✓]

Where to copy DLL: Do not copy

Run-time support: Full run-time engine

[ ] Embed project .UIRs

[ ] Generate map file

LoadExternalModule Options

[ ] Enable LoadExternalModule

Exports

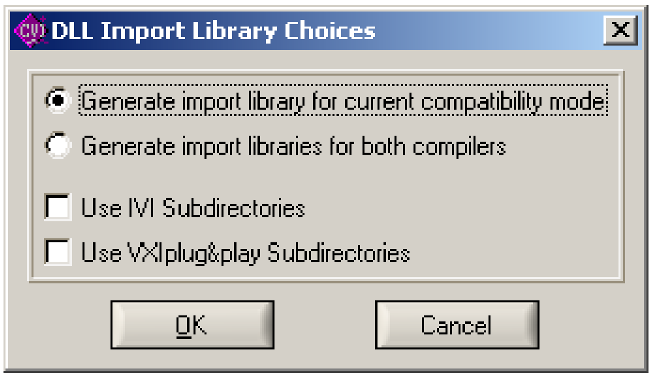
Export what: Symbols marked for export

Figure 9-15 Target Settings dialog

Build...Create Release Dynamic Link Library starts the building process for the DLL. The DLL file is built in the project directory. Note that the necessary files (DLL, LIB, CDD) must be copied manually to the Bin directory after the build.

How to complete the Version Info dialog box is shown in Figure 9-9 in section 9.5.4.6 above.

In Import Library Choices, check the current compatibility mode (which must be Microsoft Visual C/C++). Only a single LIB file is generated with this option.

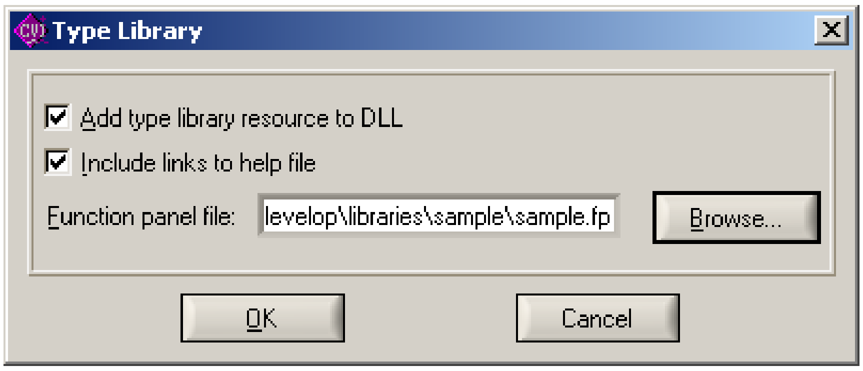

DLL Import Library Choices

(● Generate import library for current compatibility mode)
( ) Generate import libraries for both compilers

☐ Use IVI Subdirectories
☐ Use VXIplug&play Subdirectories

Figure 9-16 DLL Import Library Choices

The Type Library dialog must be completed like shown in the following figure. Adding the type library resource to the DLL enables TestStand to access the function names and prototypes in the DLL. The links to the help file enables TestStand to show function help for each function in the DLL. The path of the function panel file is the same as the path for the project. Note that the FP file must be copied to the Bin directory manually.


Type Library

☑ Add type library resource to DLL
☑ Include links to help file

Function panel file: levelop\libraries\sample\sample.fp

Figure 9-17 Type Library

The Add Files To DLL dialog is not required for build of the high-level library.

The Export Options are set to Symbols Marked for Export. All library functions marked with DLLEXPORT are exported from the DLL.

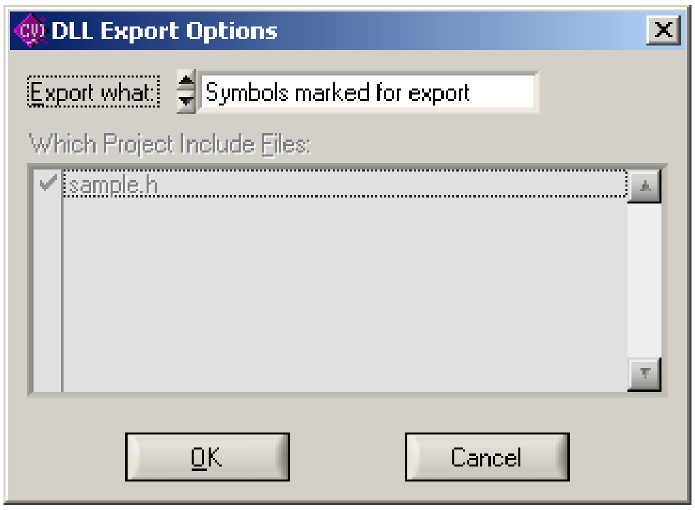
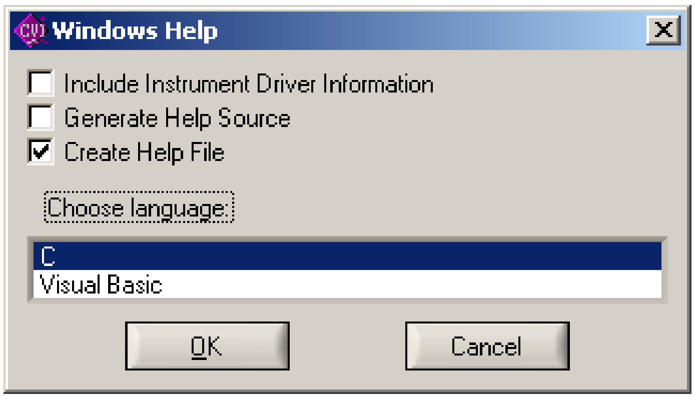


Figure 9-18 DLL Export Options

9.5.5.5 Building Help

Open the FP file for the high-level library and apply the command

Options ... Generate Documentation ... Windows Help.


Windows Help

- [ ] Include Instrument Driver Information
- [ ] Generate Help Source
- [x] Create Help File

Choose language:

C
Visual Basic

Figure 9-19 Generate Windows Help

Check the 'Create Help File' option and select 'C' for the language. After pressing the OK button, the HLP file is created from the information in the FP file. Note that the HLP file must be copied to the BIN directory manually.

17th Issue 08.09

9.6 SAMPLE project

The SAMPLE project shows how a library interacts with the resource manager during setup, measurement and cleanup. It consists of a sample sequence and a CVI project which creates a DLL. The C source code contains comments for each step and may be used as a framework to build a high-level library.

The sample project is available in the following path:
`...\Gtsl\Develop\Libraries\Sample`

10 Creation of UUT test libraries

☞

NOTE:

Knowledge of C programming is needed to create UUT test libraries.
Do not modify the original UUT Library project. Instead, make a copy of the uutlib directory and make your modifications in the copy.
Be careful when you put your private DLLs in the `gtsl\bin` directory. Always keep a copy, because they may be overwritten by an update to R&S GTSL if there is a DLL with the same name. It is safer to keep your projects and DLLs in a completely separate location beside the R&S GTSL tree. Be sure to add the directory where your DLLs reside to the PATH environment variable of your computer.

10.1 Scope

10.1.1 Identification

This chapter describes how to write a UUT Library for the R&S GTSL software.

10.1.2 System overview

The UUT Library is a high-level library which offers functions to control a mobile (UUT) through its bottom connector. These functions are called from the TestStand sequence.

The UUT Driver is the device driver for the UUT which uses a serial COM port to communicate with the mobile in most cases.

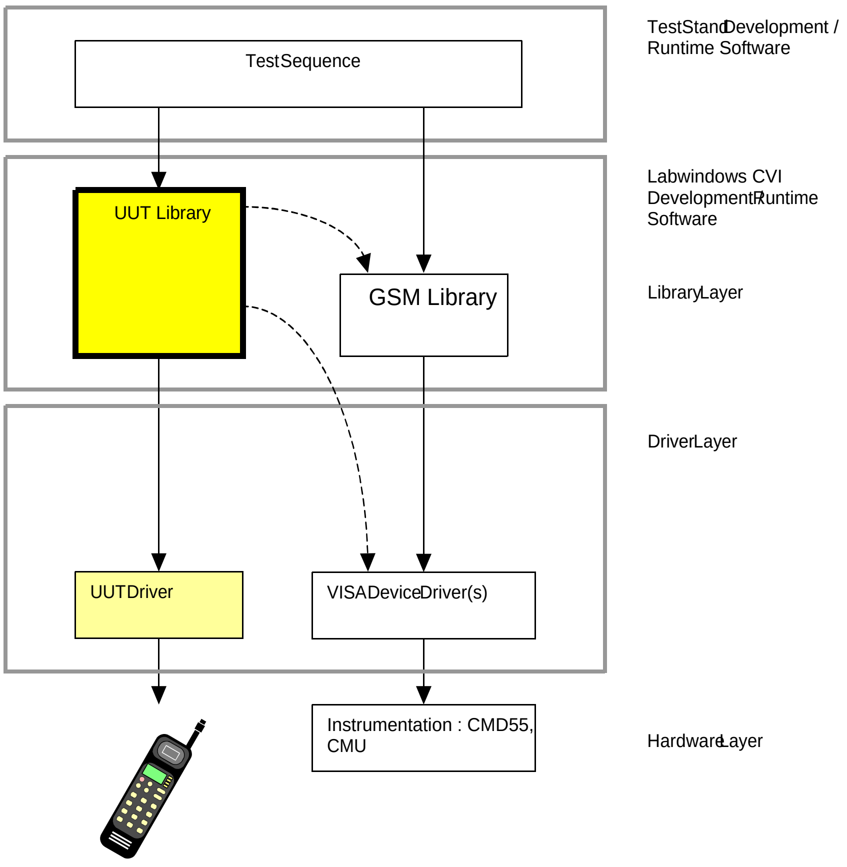

TestSequence
UUT Library
GSM Library
UUTDriver
VISADeviceDriver(s)
Instrumentation : CMD55, CMU
TestStandDevelopment / Runtime Software
Labwindows CVI DevelopmentRuntime Software
LibraryLayer
DriverLayer
HardwareLayer

Figure 10-1 UUT Library and GSM Library

The UUT Library calls functions from the UUT Driver. It may also call functions from the GSM or other high-level library (optional, project-specific) or may call the device driver directly (possible, but not recommended).

10.2 Referenced documents

[INSTR]	LabWindows/CVI Instrument Driver Developers Guide, National Instruments, February 1998 Edition
[RESMGR]	Resource Manager online help file (resmgr.hlp)
[UUTLIB]	UUT Library online help file (uutlib.hlp)

10.3 Design decisions

The 'UUT' software modules described here contain code which is specific for each mobile model, chipset etc. and therefore cannot be part of the generic R&S GTSL software. These software components are to be developed for each project and integrated into the R&S GTSL software environment.

The UUT Driver is a low level interface to the hardware, which is already present in most cases. It is normally a static or dynamic link library supplied by the mobile phone or chipset development engineers. The functions and parameters of different UUT Drivers may vary widely.

The UUT Library is a high-level library which can be kept independent of the underlying UUT Driver. It contains a basic set of UUT functions like accepting a call and an optional set of customer-specific functions like the calibration of a mobile. A template for a UUT Library containing the basic functions is part of the R&S GTSL software package.

10.4 Architectural design

10.4.1 Components

There are two CSCI components:

- The UUT Driver
- The UUT Library

10.4.1.1 UUT Driver

The UUT Driver is a piece of software, which already exists in most cases. It offers a set of functions to activate the built-in commands of the mobile chipset using the bottom connector and a serial interface. The call interface of the UUT driver will vary from case to case and must be considered 'as is'.

10.4.1.2 UUT Library

The UUT Library is a R&S GTSL high-level library and therefore conforms to the rules described in section 8.

The UUT Library uses the Resource Manager functions to gain access to the configuration description in the physical and application INI files. It uses the resource ID as a parameter to identify its resources (i.e. the bench where the UUT is defined).

It offers a set of basic functions to control the mobile:

- mobile commands (select network, start a call, receive a call)
- mobile test functions (set test mode, continuous transmission)
- measurement function (get RSS value)

It optionally contains customer-specific functions where the mobile and other high-level libraries (e.g. GSM, CDMA) are involved:

- measure transmission parameters
- calibrate mobile

The UUT Library adds resource manager capabilities like configuration description, locking/sharing, to the UUT Driver functions in the same way as a high-level library does to a device driver.

10.4.2 Concept of execution

10.4.3 Interface design

10.4.3.1 Interface identification and diagrams

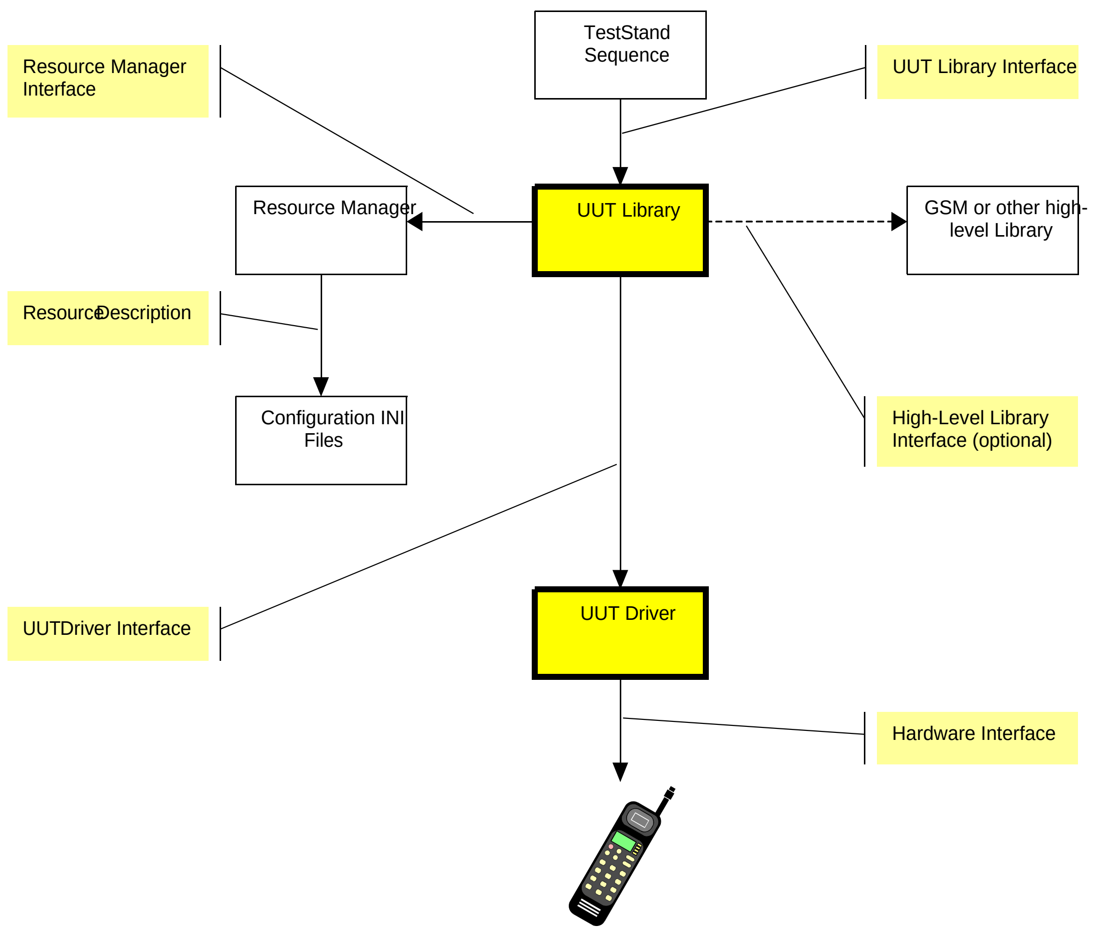


Figure 10-2 Interface diagram

10.4.3.2 UUT Library Interface

The UUT Library Interface is a standard interface for a high-level library (see chapter 9: Creation of test libraries). It presents the UUT functions at the TestStand level. See chapter 10.5.2 for a complete function reference.

10.4.3.3 UUT Driver Interface

The UUT Driver Interface is defined by the UUT Driver and cannot be influenced in most cases.

10.4.3.4 Hardware Interface

The hardware interface is defined by the UUT mode, chipset etc. Usually it is a serial interface.

10.4.3.5 Resource Manager Interface

The Resource Manager Interface is described in [RESMGR].

10.4.3.6 Resource Description

The Resource Description can be found in the physical and application INI files, see [RESMGR] and section 10.5.3.

10.4.3.7 High-Level Library Interface

The high-level library interface is defined by the respective library like GSM, CDMA etc. This interface conforms to section 8. See the online help file for a complete function reference. This interface is not required for the basic set of UUT library functions.

10.5 Detailed design

10.5.1 Coding rules

See chapter 9.5.1

10.5.2 UUT Library reference

10.5.2.1 Setup function

See chapter 9.5.2.1

10.5.2.2 Cleanup function

See chapter 9.5.2.2

10.5.2.3 Library version function

See chapter 9.5.2.3

10.5.2.4 Set resource ID function

Purpose: Pass the resource id of a high-level library along with a descriptor string to the UUT Library. The resource id is used by the UUT Library to call the high-level library in complex customer-specific functions (not used for the basic functions).

```
Prototype UUTLIB_Set_Resource_Id
      ( CAObjHandle  seqContext,
        long         resourceId,
        long         extResourceId,
        char       * extResourceTag,
        short      * errorOccurred,
        long       * errorCode,
        char         errorMessage[] );

long     extResourceId :
resource ID of the high-level library

char   * extResourceTag :
describes how to use the resource id. Normally the
name of the high-level lib for the resource id like
'GSM' or 'CDMA'.
```

10.5.3 Resource description

The resource descriptor for the UUT Library is the name of a bench. The bench device 'UUT' must be present.

Instead of a bench name, a logical name pointing to a bench may also be used.

The following keys are supported by the UUT Library:

Key name	Remarks
Simulation	if set to '1', the complete bench is simulated by the library (mandatory, see 8.5.3)
Trace	if set to '1', tracing is enabled for the library (recommended, see 8.5.3)
UUT	bench device, the link to the UUT device entry (mandatory)

Table 10-1 Bench properties

Key name	Remarks
Type	Device type, project-specific (mandatory). See remarks below.
ResourceDesc	VISA resource descriptor like 'GPIB0::15' or 'PXI0::16::0'. The given value must be passed to the device drivers init function. Non-VISA descriptors like 'COM2' may be used for serial interfaces (mandatory).

Table 10-2 Device properties

The template UUT library already supports three different 'Type' entries:

- Type = 'DEMO': Each function of the UUT Library displays a dialog box when executed and simulation mode is off.
- Type = 'USER': The UUT library contains 'TODO' comments where code for a customer-specific UUT driver has to be inserted. The keyword 'USER' may be easily replaced by a more meaningful one.
- Additional types may be added easily.

Additional properties may be defined if required (project-specific). Think about baud rate, parity settings etc. of the serial interface which could be set in the physical or application ini file rather than hard-coding them in the UUT library or the driver.

Example:

application layer:

```
[LogicalNames]
GSM = bench->GSM

[bench->GSM]
RadioComTester = device->CMU
UUT = device->gsm_mobile
Simulation = 0
Trace = 1

[device->gsm_mobile]
Type = USER
```

physical layer:

```
[device->gsm_mobile]
Type = DEMO
ResourceDesc = COM2
```

In the above example, the following resource names can be used in the UUTLIB_Setup function:

'bench->GSM'	(bench name)
'GSM'	(logical name for the bench)

Note that the [device->gsm_mobile] entry exists in both layers. The physical layer defines the hardware interface COM2 for the communication. The mobile type is set to 'DEMO' here. The application layer overwrites the 'DEMO' type with the appropriate model type 'USER' for the application. So, it is possible to define different models in different application layer INI files.

10.5.4 UUT Driver

The interface of the UUT Driver is project-specific. The driver may exist as a source file or a library. In the first case, the source may be included in the UUT Library CVI project, in the second case the object or library file must be linked into the project.

10.5.5 Miscellaneous

10.5.5.1 Support of several UUT Drivers in one library

There are two ways to handle a set of UUT Libraries and Drivers for different mobile models. The simplest approach is to have separate UUT Libraries and drivers with different file names for each model. The selection of the UUT model is done by using the appropriate name of the DLL files for the UUT library in the test sequence:

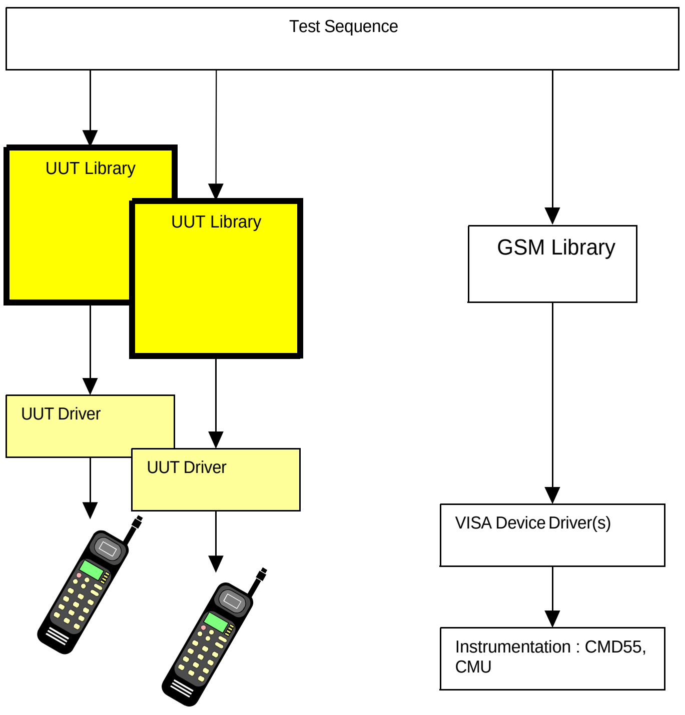


Figure 10-3 Separate library for each mobile model

A second approach could be a common UUT Library which calls the appropriate UUT Drivers depending on the 'Type' entry in the resource description. The template UUT library is based on this concept and contains already the code to switch between the two types 'DEMO' and 'USER'.

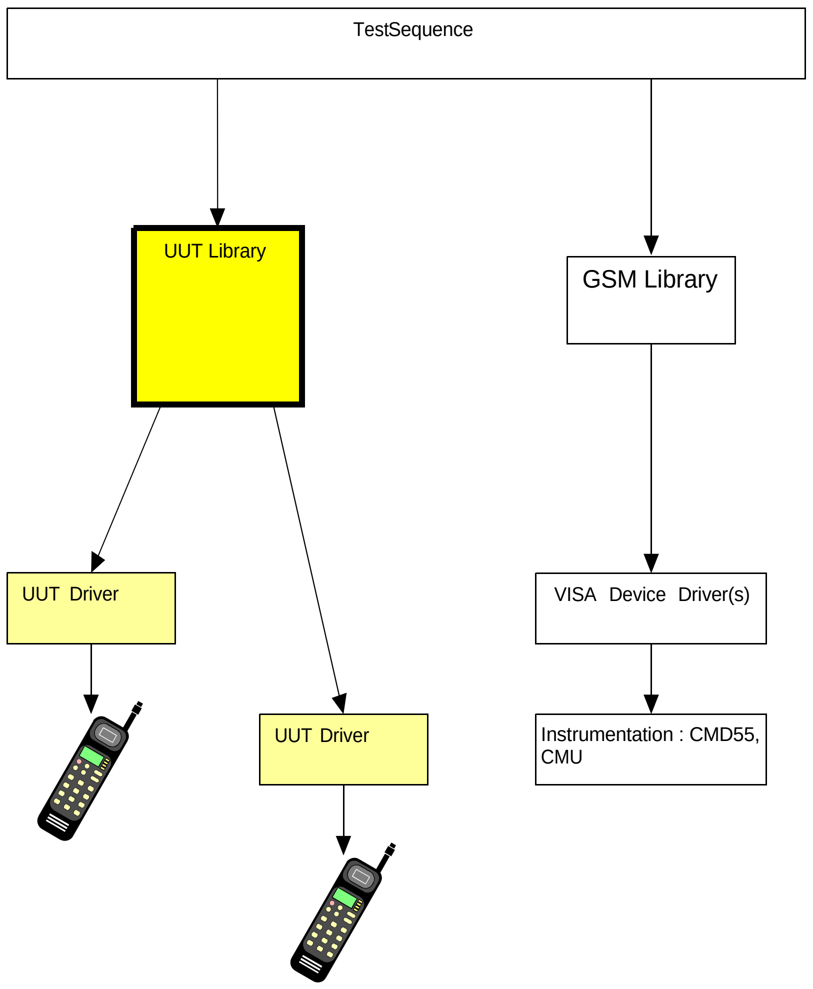


Figure 10-4 Driver selection by UUT Library

Discussion:

Separate Library for each Model:

- Good solution if the UUT Library and the TestStand sequence is significantly different for each model.

Driver Selection by UUT Library:

- Good solution if the UUT Library is the same or very similar for all models.
- Problem: The UUT Library loads all drivers as soon as it is loaded into memory. That means that always all drivers must be present on disk and that memory is wasted.
- If the driver DLLs are loaded dynamically (LoadLibrary, GetProcAddress etc.), the problem no longer exists, but the UUT Library becomes more complex.

10.5.5.2 Static versus Dynamic Linkage

The UUT Library and UUT Driver modules may be linked together in a static or a dynamic way. There are three ways in which the modules can be linked:

10.5.5.2.1 Static linking

If the source code of the UUT Library and the UUT driver is available, all these C files can be included in a single CVI project. The files are compiled and linked into a single output DLL. For each UUT type, there is a DLL with a different name. The function names inside the DLLs may be the same (like UUTLIB_Setup) or may be UUT specific (like UUT_MODEL_XYZ_Setup). There must be a separate TestStand sequence for each UUT model, since the name of the UUT Library is different for each model.

Code may be re-used only by making a copy of the source files from an earlier project.

If the UUT driver is not available as source code, only as an object file (xyz.OBJ) or a static library (xyz.lib), the result is the same. There is a single DLL containing the executable code for the UUT library and the driver:

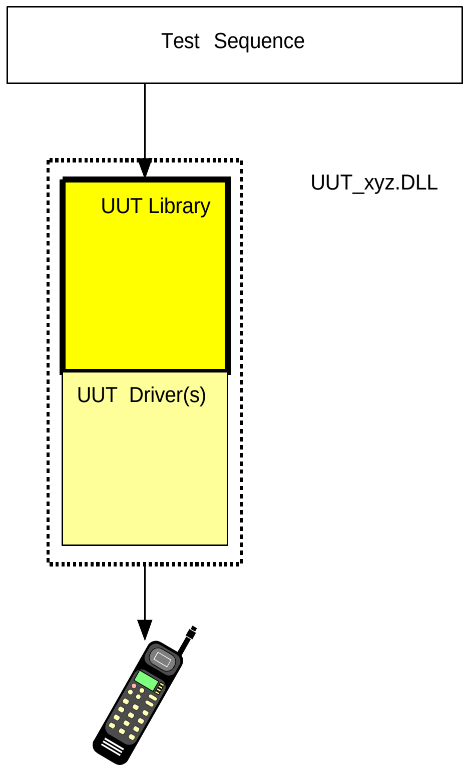


Figure 10-5 Static linking

10.5.5.2.2 Load-time dynamic linking

If the UUT driver comes as a DLL file with an import library (LIB), the LIB file must be added to the CVI project for the UUT Library.

When the UUT Library DLL is loaded from the TestStand Sequence, the linked driver DLL is also loaded into memory. Loading will fail if the dependent DLL cannot be found in the PATH environment variable. The names of the DLL to be loaded is defined at compile/link time of the project and cannot be changed at run-time.

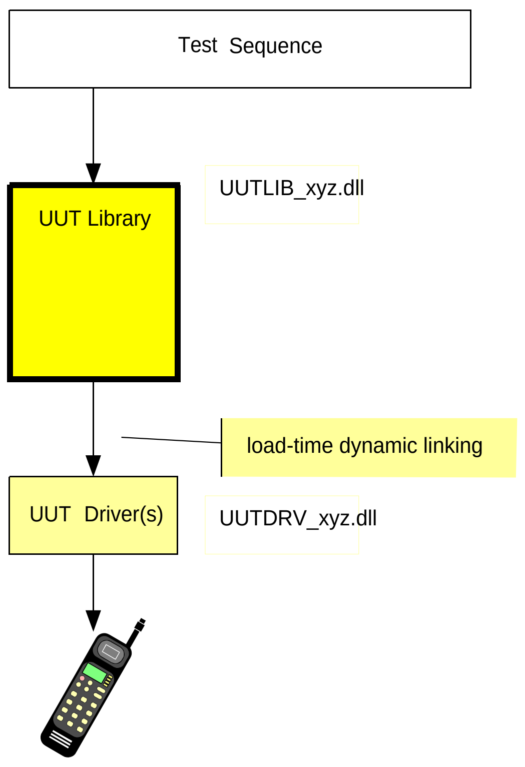


Figure 10-6 Load-time dynamic linking

A problem may arise if a UUT Library DLL links with several UUT driver DLLs to support several UUT types. As soon as the UUT Library DLL is loaded from TestStand, it will search for all dependent driver DLLs and load them into memory. If only one of them is missing, loading the UUT Library fails.

10.5.5.2.3 Run-time dynamic linking

Run-time dynamic linking occurs when a DLL uses the LoadLibrary and GetProcAddress functions to get the starting address of a DLL function. This eliminates the need to link with an import library. The target DLL is loaded only when LoadLibrary is called with the appropriate file name.

If a UUT Library DLL links with several UUT driver DLLs using run-time dynamic linking, it is not necessary that all dependent driver DLLs are present. Only the DLL required for the specific UUT type is loaded at run-time with LoadLibrary.

The disadvantage of this approach is the complexity of the source code. Consider the following example code for a UUT Library:

Load-time dynamic linking

```
// Driver function from the import library ABC.LIB
extern int ABC_Close (long Handle, char *ErrorMsg);

int result;
long h_uut;
char errmsg[2048];
...
result = ABC_Close ( h_uut, errmsg );
```

The ABC_Close function is usually declared in the include file of the UUT Driver, in this case ABC.H. The import library ABC.LIB is linked with the UUT Library. When the UUT Library is loaded, the corresponding ABC.DLL will be loaded too.

Run-time dynamic linking

```
// function prototype
typedef int (*CLEANUP_FUNCTION)(long li_Handle, char *pc_ErrorMsg);

HINSTANCE hinst_lib;
CLEANUP_FUNCTION cleanup_fct;
int result;
long h_uut;
char errmsg[2048];

hinst_lib = LoadLibrary ('ABC.DLL');   // Load the DLL
if ( hinst_lib != NULL )
{
   // find the procedure address using the procedure name
   cleanup_fct = (CLEANUP_FUNCTION) GetProcAddress ( hinst_lib, 'ABC_Close' );
   if ( cleanup_fct != NULL )
      result = (cleanup_fct) ( h_uut, errmsg );
   else
   {
      // error handling, could not find driver function
   }
}
else
{
   // error handling, could not load driver DLL
}

FreeLibrary ( hinst_lib );
```

The prototype of the cleanup function is declared in a 'typedef' statement at the beginning. Before the function can be called, the DLL must be loaded with LoadLibrary and the name of the DLL. The load process searches the PATH environment variable for a DLL with this name and loads it into memory. Loading the library is usually done only once in the UUTLIB_Setup function. The same applies to the FreeLibrary function at the end, it is called from UUTLIB_Cleanup. The name of the DLL is a string parameter to LoadLibrary, so it is easy to support any number of UUT Drivers and use the 'Type' entry of the application INI file as the library name.

Before the function can be called, the procedure address must be retrieved with the function GetProcAddress. The name of the function is a string parameter, so again we can distinguish between the different libraries using the 'Type' as a prefix for the function name.

Note that all functions of a driver DLL must be called using the GetProcAddress method. As soon as you include only one explicit function call (as with load-time dynamic linking), the driver DLL is loaded at load-time.

10.5.5.3 Adding complex UUT Library functions

The UUT Library template offers a basic set of UUT functions. These functions call the UUT Driver. Interactions between the mobile and the radio communication tester are normally done by calling the UUT and the GSM (or other high-level) library alternately from the TestStand sequence.

However, these interactions sometimes may become quite complex, like the calibration of a mobile. In this case, it makes sense to write this procedure in C code rather than in a TestStand sequence. Since this procedure depends on the UUT, it is best to include it in the UUT Library.

The UUT Library must be able to call functions from other high-level libraries like the GSM library. In some special cases it may even be necessary to call the driver of the radio communication tester directly:

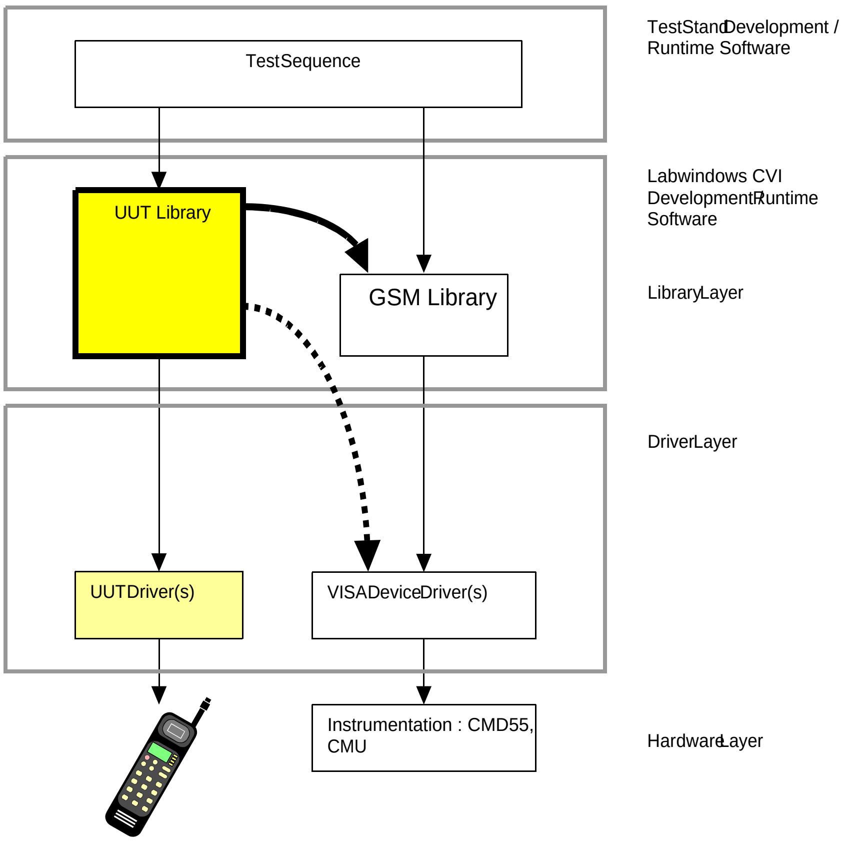

TestStand Development / Runtime Software
TestSequence
Labwindows CVI Development/Runtime Software
UUT Library
GSM Library
Library Layer
Driver Layer
UUT Driver(s)
VISA Device Driver(s)
Instrumentation : CMD55, CMU
Hardware Layer

Figure 10-7 Complex UUT Library function calls

When the Setup functions of the GSM Library and UUT Library are called, each of them returns a resource id. Both resource ids are stored in the test sequence. For subsequent function calls of the two libraries, the appropriate resource id must be passed to the function.

When we want to call the GSM Library from the UUT Library, we must know the resource id for the GSM Library inside the UUT Library. We cannot pass the resource id of the UUT library to the GSM library.

The UUT Library offers the function UUTLIB_Set_Resource_Id where the resource id of an external library is passed to the UUT Library along with a resource tag (a string, which identifies the external library). The UUT library analyzes the tag and stores the external resource id in its memory block associated with the resource id. After that, the complex function which calls the GSM library is able to retrieve the appropriate resource id for the GSM library from the memory block. When a function needs to deal with more than one external library, the UUTLIB_Set_Resource_Id function has to be called for each external resource ID with the appropriate tag.

10.5.6 The UUT Library template

10.5.6.1 Directory structure

The template for the UUT Library can be found in the subdirectory \develop\libraries\uutlib in the R&S GTSL tree. It consists of the CVI project file and source files for a UUT Library with basic functionality.

10.5.6.2 Customizing the UUT Library

☞

NOTE:

Do not modify the original UUT Library project. Instead, make a copy of the uutlib directory and make your modifications in the copy.
Be careful when you put your private DLLs in the `gtsl\bin` directory. Always keep a copy, because they may be overwritten by an update to R&S GTSL if there is a DLL with the same name. It is safer to keep your projects and DLLs in a completely separate location beside the R&S GTSL tree. Be sure to add the directory where your DLLs reside to the PATH environment variable of your computer.

10.5.6.2.1 Copy the project

First of all, make a copy of the files in the \develop\libraries\uutlib in a separate location before you start your work.

When you load the CVI project, some warnings may appear, telling you that the project has been moved and some files could not be found. Add the `gtsl\include` directory to the list of include paths and replace the LIB files that could not be found by the LIB files from the `gtsl\bin` directory. Now the project is ready to be compiled and linked.

10.5.6.2.2 Add the UUT Driver files

Copy the necessary files for your UUT driver to the project directory and include them in the CVI project. Usually, this is the H file with the external function definitions and a C, OBJ or LIB file.

10.5.6.2.3 Define the UUT type

The standard UUT type is 'USER'. You may replace this string by a more meaningful name for your mobile model or add a new type.

UUTLIB.H defines a set of strings for the known device types UUTLIB.C and an integer number for each type. There is also a special number UUTLIB_TYPE_INIT which means that no type has been defined yet.

Code that has to be added for the new type 'MyMobile' is shown in *italics*.

```
#define UUTLIB_TYPE_DEMO_STRING 'DEMO'
#define UUTLIB_TYPE_USER_STRING 'USER'
#define UUTLIB_TYPE_MYMOBILE_STRING 'MyMobile'

#define UUTLIB_TYPE_INIT               0
#define UUTLIB_TYPE_DEMO               1
#define UUTLIB_TYPE_USER               2
#define UUTLIB_TYPE_MYMOBILE           3
```

At the beginning of the UUTLIB_Setup function, the UUT type is set to 'undefined':

```
    /* set default values */
    bench_ptr -> type_uut = UUTLIB_TYPE_INIT;
```

Later on, the value of the 'Type' key from the device section is compared against the predefined type strings and one of the type constants is assigned to bench_ptr->type_uut. If no matching entry can be found, the function returns an error:

```
if( !CompareStrings( buffer, 0, UUTLIB_TYPE_DEMO_STRING, 0, 0 ) )
  {
      bench_ptr -> type_uut = UUTLIB_TYPE_DEMO;
  }
  else if( !CompareStrings( buffer, 0, UUTLIB_TYPE_USER_STRING, 0, 0 ) )
  {
      bench_ptr -> type_uut = UUTLIB_TYPE_USER;
  }
  else if( !CompareStrings( buffer, 0, UUTLIB_TYPE_MYMOBILE_STRING, 0, 0 ) )
  }
      bench_ptr -> type_uut = UUTLIB_TYPE_MYMOBILE;

   /* **TODO** insert additional CompareStrings' if more types are supported */

   }
   else
   {
      /* no supported type key entry found */
      *errorOccurred = 1;
      *errorCode = UUTLIB_ERR_NO_SUPP_TYPE;
      format_error (errorMessage, *errorCode, *ptr_resourceID, RESMGR_KEY_UUT);
      goto Error;
   }
}
```

Each UUT Library function has a switch/case statement for the device type. You have to add an additional case for the new device type. In case of an unknown type, the function returns an error through the 'default' case:

```
switch( bench_ptr->type_uut )
  {
   case UUTLIB_TYPE_DEMO:
      RESMGR_Trace ( 'UUT Driver: Initialize UUT' );
      MessagePopup('UUT Driver','Initialize UUT');
      session_handle = UUTLIB_DUMMY_HANDLE;
      break;

   case UUTLIB_TYPE_USER:
      /* **TODO** insert here UUT initialization routines */
      break;

   case UUTLIB_TYPE_MYMOBILE:
      /* **TODO** insert here UUT initialization routines */
      break;

   /* ------------------------------------------------------------ */
   /* ------------------------------------------------------------ */
   /* **TODO** insert additional cases if other types are supported*/
   /* ------------------------------------------------------------ */
   /* ------------------------------------------------------------ */


   default:
      *errorOccurred = 1;
      *errorCode = UUTLIB_ERR_NO_TYPE_PROC;
      format_error (errorMessage,*errorCode,*ptr_resourceID,RESMGR_KEY_UUT);
      goto Error;

}
```

10.5.6.2.4 Add the UUT Driver function calls

Add a #include line for the UUT driver's H file.

Search for the 'TODO' comments and add the appropriate calls to the UUT driver functions.

10.5.6.2.5 Support for additional drivers

Note that the procedure names of all driver modules must be unique throughout the CVI project.

Copy the files for all drivers to the project directory and include them in the CVI project.

Add a UUT type for each driver.

Add a case clause for each new type in each UUT library function and add the necessary driver calls.

11 Creation of self test libraries

NOTE:

Knowledge of C programming is needed to create self test libraries.
Do not modify the original Sample Self Test Library project. Instead, make a copy of the sftcsample directory and make your modifications in the copy.
Be careful when you put your private DLLs in the `gtsl\bin` directory. Always keep a copy, because they may be overwritten by an update to R&S GTSL if there is a DLL with the same name. It is safer to keep your projects and DLLs in a completely separate location beside the R&S GTSL tree. Be sure to add the directory where your DLLs reside to the PATH environment variable of your computer.

11.1 Scope

11.1.1 Identification

This guide describes how to write a self test library for the R&S GTSL software.

11.1.2 System overview

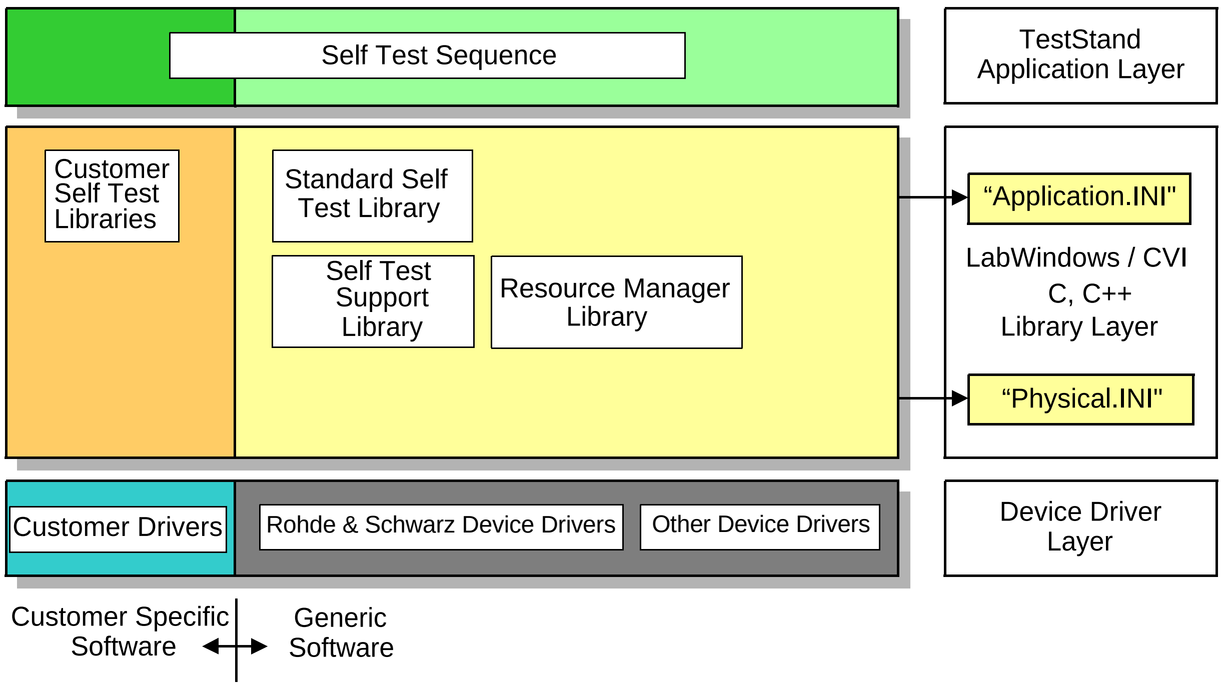


Figure 11-1 R&S GTSL Software Overview

A self test library offers a group of functions which cover the needs for testing a specific device or some equipment in a test system.

The library functions are called from a TestStand sequence. The functions themselves interact with the resource manager library, the self test support library and the device driver(s).

This chapter describes, how a self test library interacts with the R&S GTSL software and how to write such a library.

11.2 Referenced documents

[RESMGR]	Resource Manager online help file (resmgr.hlp)
[SFTSUP]	Self Test Support Library online help file (sft.hlp)
[INSTR]	LabWindows/CVI Instrument Driver Developers Guide, National Instruments, February 1998 Edition

11.3 Overview

The design of any R&S GTSL self test library must meet the following requirements:

- The library must be delivered as a Dynamic Link Library (DLL), including type library information and a function panel
- The name of a system self test library starts with ”sfts”
- The name of a customer self test library starts with ”sftc”
- The function call interface must follow the R&S GTSL template.
- The library must use the resource manager functions (if applicable).
- The library must use the self test support library functions (if applicable).

These requirements are described in detail in the following sections.

11.3.1 Test System Configuration

The self test of a production test system based on the TSVP platform must be able to verify the correct functionality of the complete system. The self test library must be able to identify and report a defective part or component in the system. A part may be a device (like a CMU or a power supply), a cable (connecting a CMU to a relay card or to the fixture), a fixture or any other component inside the system (a serial or parallel interface) or outside the system (e.g. a barcode reader). A component may be a built-in card (like a relay card R&S TS-PRL1 or a PXI multimeter card) or an option in a device like CMU.

The actual configuration of the production test system can vary in a wide range. The only common part of all systems is the TSVP frame:

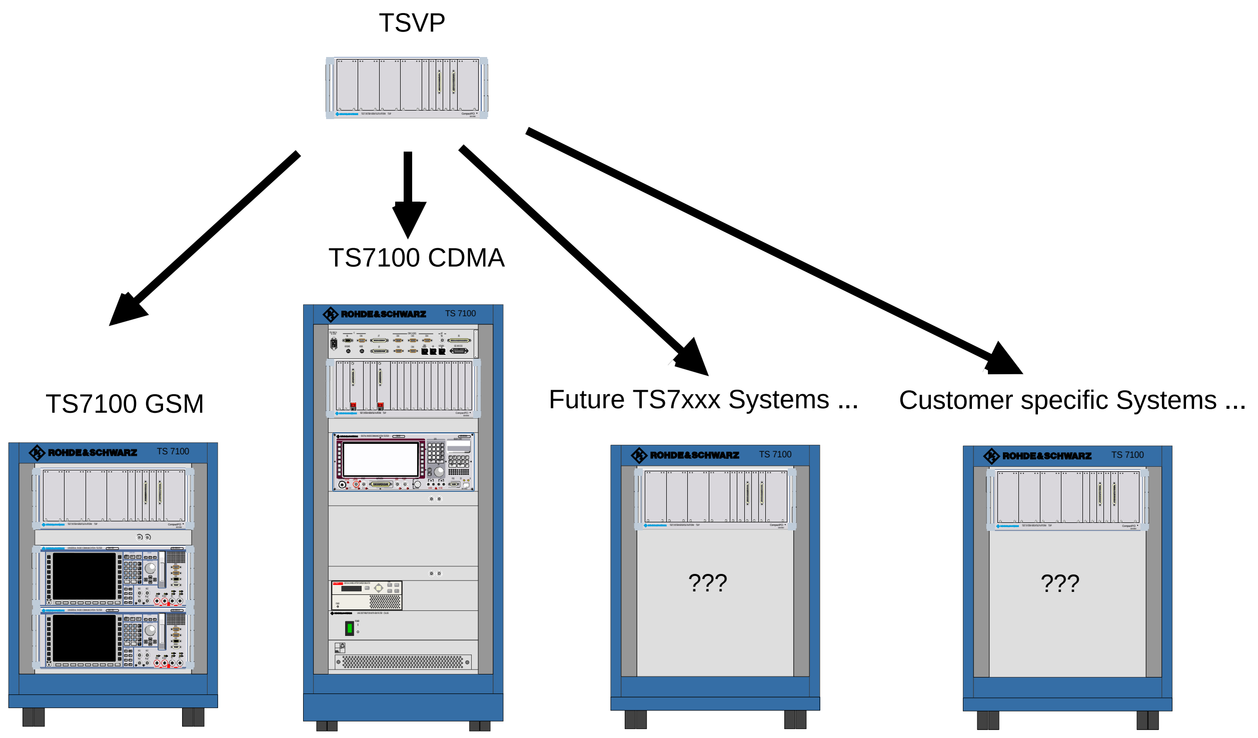


Figure 11-2 Production Test Systems based on the TSVP Platform

The concept for the TSVP self test provides functions to identify the testable components and to test these components. This can be done easily because there are no cross-tests between the components. Each component can be tested independent on the others, and the test is well defined. The self test software of a TSVP standard component is not open to the user, because there is no need to modify it.

On the other hand, the system and overall self test must be open to the user, because we do not known today, how each system will look at the customer site tomorrow. If a test system is modified and expanded by a system integrator, he is also responsible to supply the corresponding self test software part for it.

11.3.2 Self Test Levels

There are several levels of self test:

The module self test ensures that a module (e.g. a CPCI card) inside the TSVP frame is working well. The only resources for the module self test are the module under test itself, a self test board on the front connector of the module (if necessary) and the TSVP system/self-test instrumentation. A module self test will be supplied for each card developed by Rohde & Schwarz (e.g. R&S TS-PRL1, R&S TS-PMA etc.).

The system self test consists of the TSVP self test and further tests, including the external devices (e.g. GPIB bus devices), the cabling between the devices and the TSVP frame. The system self test is specific to a standard test system like the TS7100 GSM. The system self test may use any resource which has been tested before. This is necessary to perform the cabling test.

The overall self test consists of the system self test and includes tests for customer-specific extensions and modifications of the system like fixture tests, environment tests (e.g. external interfaces, barcode readers, line integration etc.)

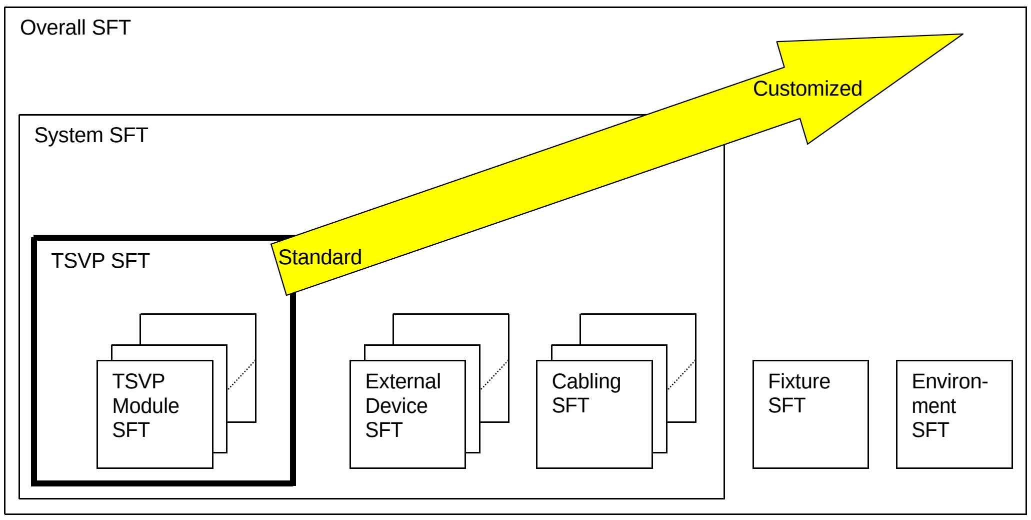


Figure 11-3 Self Test Levels

The inner levels are independent from the actual system configuration to a great extent, that means that the TSVP module self test will run on all systems. The outer levels are very system-specific and have to be customized for each system.

11.4 Software architectural design

11.4.1 Software Components

The basic components of each self test library are:

• sftxyz.h	include file
• sftxyz.lib	import library
• sftxyz.dll	dynamic link library
• sftxyz.fp	LabWindows CVI function panel
• sftxyz.c	source file
• sftxyz.prj	CVI project file
• ...	additional source/include files (if applicable)

11.4.2 Concept of Execution

The basic self test concept is not very different from any other test application. There is a test sequence in TestStand, there is a set of DLLs where the test cases are coded, there is the Resource Manager which coordinates the actions and there are the device drivers which connect the software to the devices. This makes it easy for the user, since there is no difference between loading and running a test program and running the self test. It makes it easy for the programmer, since the self test libraries are written the same way as the high-level test libraries.

The self test is configured by entries in the Resource Manager's physical and application INI files. The physical layer describes the devices, the application layer contains information about the parts to test, the self test benches and options.

11.4.2.1 Self Test Sequence

The self test sequence calls functions from the standard and customer self test libraries:

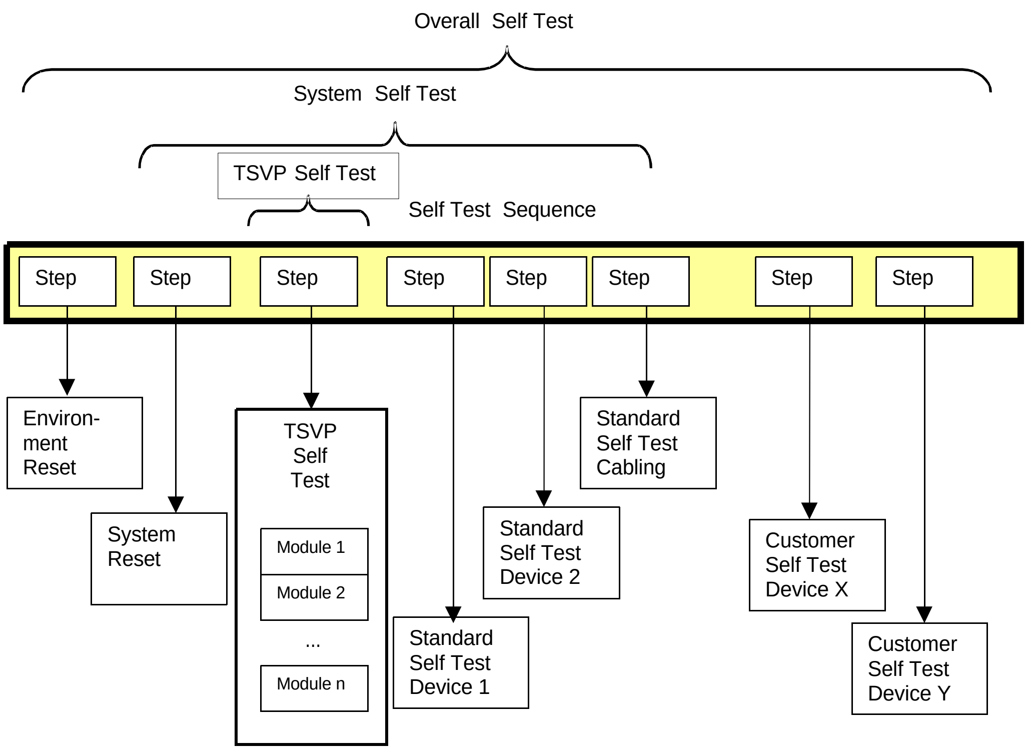


Figure 11-4 Self Test Sequence

What is the difference between a function in a self test library and a function in a device driver? Why don't we call the device driver from the TestStand sequence directly? There is a number of benefits using a self test library instead of calling a device driver directly:

- the library can handle more than one device (bench concept)
- the library can switch between different types of devices without modification of the TestStand sequence (e.g. GSM library: substitution of a CMD55 with a CMU)
- standard INI-file concept for resource description (physical/application layer)
- standard error handling mechanism, suited for use with TestStand
- the library can handle the provided functions of the self test support library more effective

The required functionality is provided by the Resource Manager library (see [RESMGR]) and the self test support library (see [SFTSUP]).

These libraries are the central parts of the R&S GTSL self test software. They coordinate the interaction between all the self test libraries. Therefore it is mandatory to use them in each self test library.

11.4.2.2 TSVP Self Test

The TSVP self test is completely kept outside the self test sequence, because it may become quite complex. The TSVP self test must identify all testable modules, load the module SFT libraries and call the appropriate test functions. This task is better implemented in C, because TestStand is not suitable. The TSVP self test consists of the TSVP Self Test Frame and a TSVP Module Self Test for each type of hardware module.

This part of the self test is not a subject of this document.

11.4.2.3 Standard Self Test Libraries

The Standard Self Test Libraries offer functions for testing standard devices and cabling. Each function is called directly from the self test sequence.

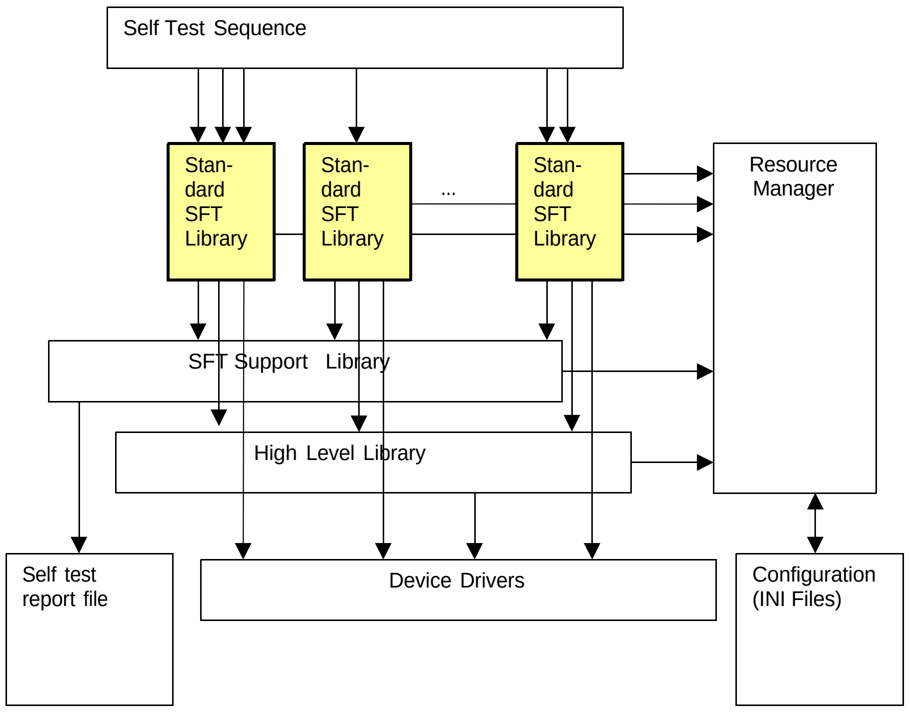


Figure 11-5 Standard Self Test Libraries

Standard self test functions are grouped into several libraries. The functions communicate with the self test support library, the Resource

Manager and the device drivers. These libraries are very similar to the "high-level libraries" described in chapter 9. A self test library may also call any other high level library (like the switch manager) to perform it's task.

11.4.2.4 Customer Self Test Libraries

Customer Self Test Libraries are implemented the same way as standard self test libraries, except that they are written by the customer or an integration center. These libraries contain tests for "non-standard" system extensions.

11.4.2.5 Self Test Support Library

The self test support library contains common functions for all other self test libraries, like writing to a report file, dialog boxes, basic measurements etc. (see [SFTSUP]).

11.4.2.6 Configuration Information

The self test uses the configuration information from the physical and application layer INI files. The physical layer describes the devices and device types. The application layer INI file describes the self test benches, user options and selectable self test parts.

11.4.3 Interface design

11.4.3.1 Interface Identification and Diagram

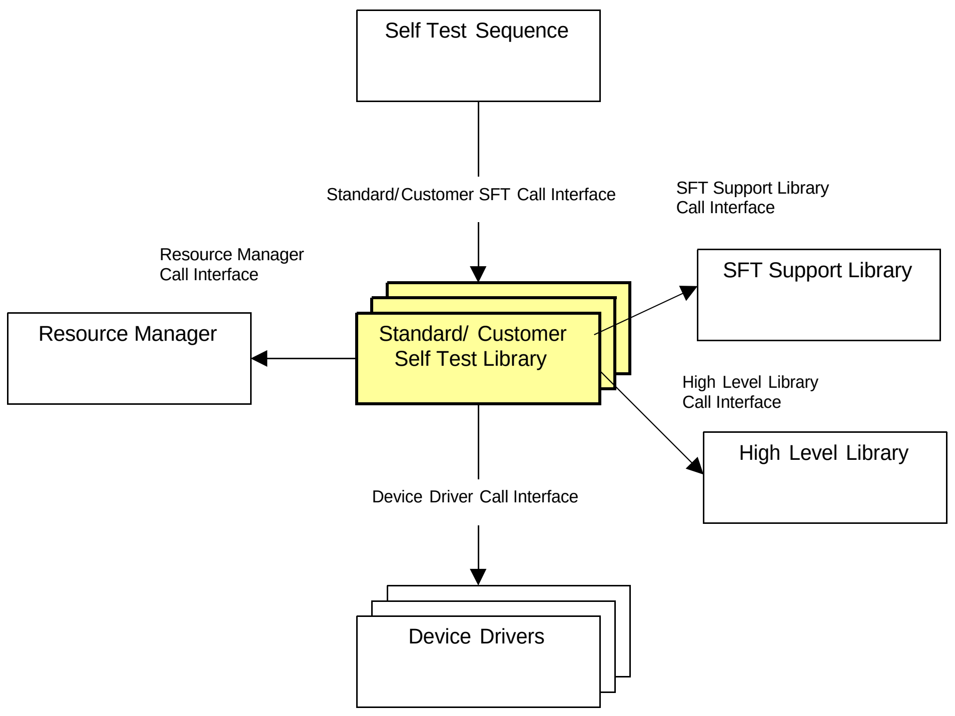


Figure 11-6 System/Overall Self Test Interfaces

11.4.3.2 Standard/Customer SFT Call Interface

The Standard/Customer SFT Call Interface conforms to the rules for the interfaces for high-level libraries and is described in chapter 9.

Because the library normally exports only a single self test function, it is legal to omit the Setup and Cleanup functions from the call interface and include them at the beginning and at the end of the self test routine. A XXX_Lib_Version() is also not necessary because the actual version of a self test library is written directly to the report file.

11.4.3.3 Resource Manager Call Interface

The Resource Manager Call Interface is described in [RESMGR]. The SFT libraries use this interface to access information from the configuration INI files and to handle resource ID's, session handles and locking.

11.4.3.4 Device Driver Call Interface

The Device Driver Call Interface is defined by the IVI, VISA or other device drivers for the modules under test.

11.4.3.5 High Level Library Call Interface

See chapter 9.4.3: Interface design.

11.4.3.6 SFT Support Library Call Interface

The SFT Support library provides a Setup and a Cleanup function. The Setup function must be called from the SFT sequence before any other function (TSVP self test, standard and customer self test function) can be called. The Cleanup function must be called at the end of the self test sequence.

The **Setup function** initializes the library and reads configuration information from the application layer and physical layer INI files. Information about the run-time state of the self test is also initialized in this function. Next, it calls the Dialog function which displays a dialog window where the user can set the self test options. A list of all available self test parts is displayed which can be selected or deselected. When the user closes the dialog window, the information is stored in the run-time state of the self test support library and all self test libraries can read that information.

A set of **Get/Set Attribute functions** is provided to access and modify the run-time state of the library. These functions can be called by the self test modules to get information about the selected SFT options and to set information about the test result (Pass/Fail).

A set of **Report functions** is provided to write the test results to the report file in a standardized form.

User Interface functions display the test progress on the screen and provide standardized dialogs (e.g. for component selection).

A set of **Measurement functions** is provided to access the basic measurement equipment for the self test like voltage, current, resistor measurements and functions to switch the DMM to the analog bus.

11.5 Software detailed design

11.5.1 Coding Rules

See chapter 9.5.1: Coding rules.

11.5.2 Self Test Sequence

The self test of the system is done by a self test sequence for National Instruments TestStand. No model is required for that sequence.

11.5.2.1 MainSequence Setup

- RESMGR_Setup loads the physical INI and the self test application INI file
- SFT_Setup initializes the self test support library and displays the self test dialog windows

11.5.2.2 MainSequence Main

- ... other standard/customer SFT library calls
- SFTSTSVP_Test performs the TSVP self test
- ... other standard/customer SFT library calls

11.5.2.3 MainSequence Cleanup

- SFT_Cleanup closes the self test support library
- RESMGR_Cleanup closes the resource manager

11.5.3 Standard and Customer Self Test Libraries Reference

11.5.3.1 Overview

There is an naming convention for the self test libraries:

- SFTSxxxx.DLL (standard)
- SFTCxxxx.DLL (customer-specific)

where xxxx identifies the system or module. The name of the DLL in uppercase characters is identical to the prefix used for every exported symbol and every internal defined constant value of the library. The prefix must also be used in the function panel file for LabWindows CVI and therefore it must contain only alphanumeric characters.

The interface and internal structure of Standard and Customer Self Test Libraries are identical. The libraries conform to the architecture described in the other chapters.

Because a self test library normally exports only a single function, it is legal to omit the Setup and Cleanup functions from the call interface and include them at the beginning and at the end of the self test routine. A XXX_Lib_Version() is also not necessary because the actual version of a self test library is written directly to the report file.

The following chapters show how a customer self test library could look like. See the source file "sftcsample.c" in the SFT sample project for details.

11.5.3.2 Self test concept

All results of a self test library function call are written in a common report file (see Sft_report.txt of the sample project). The name and location of that file can be configured in the application layer INI file and changed via dialog at runtime. The report consists of the following elements:

- parts
- components
- test cases
- test case informations
- run-time errors

All self test parts, components, test cases and test case information objects are hold in a tree structure. All items within one level must have unique names. The run – time error messages and the objects on the lowest level have no names.

```
run – time errors
- part 1
    - component 1
        - test case 1
          comment
          comment
          table
          comment
          table
    - component 2
        - test case 1
          comment
```

```
                result
                result
            + component 3
        + part 2
        + part 3
```

11.5.3.2.1 Part

Parts are defined in the application layer INI file. The information from the [SftParts] section is kept in the part list in the self test support library. This list is generated by the SFT_Setup function. A self test part may be a device in a measurement system or the cabling. Read and write access is accomplished by Get/Set Attribute functions. This functions work on the active part. There is a function to select a part. If components are added to the self test, they are associated to the selected part.

11.5.3.2.2 Component

To perform tests for a part at least one ”component” must be added. A component may be a simple device (Power Supply), a option in a device (CMU) or a plug in card in the TSVP. The SFT support library provides functions to add, select and modify component items. All component items are stored in an internal list of the self test support library. Additionally there is a dialog function which allows the user to select the components before the self test for that part is started. The programmer of the self test library can decide whether to show this dialog or not. The dialog shows all component items in the list and the user can modify the ”selected” attribute. If test cases are added to the self test, they are associated to the active component.

11.5.3.2.3 Test case

To do tests for a component the self test library has to create at least one test case item. The self test support library provides functions to add, select or modify test case items. All items are stored in an internal list of the self test support library.

11.5.3.2.4 Test case information

A self test library can associate any of the following objects to the activated test case using the report functions:

- comment
- text result

- error message
- warning
- flexible table
- measurement result table

11.5.3.2.5 Run – time error

Unexpected or severe errors are handled in the way described in chapter 9.5.4.1: Error handling. Such errors will cause TestStand to pop up a “Run – Time Error” dialog. Additionally the programmer of the self test library is responsible that this error text is written to the self test report by using the appropriate function in the self test support library.

11.5.3.3 Configuration Information

The configuration information for each self test library is similar to the configuration information for a high-level library (cf. [RESMGR]). Each library requires a bench section, where the device under test and the devices required to perform the self test are described.

Additionally the self test support library requires some entries in the application layer INI file for the SFT_Setup function. See chapter 7.1.10 for details.

To show the concept of a self test library a sample project is added to the R&S GTSL software. The entry ”SAMPLE” in section [SftParts] in the application layer INI file is created for that library.

The sample library tests two imaginary components. One is called "AUX" the other one "DCS". With the AUX component some report functions of the self test support library are shown. The DCS component is a imaginary device. To test this device it is necessary to open a session with the device driver. The ports DCS_HI and DCS_LO of the device DCS are connected to the Self Test Matrix Card via channel P11 and P13. The Switch Manager is used to perform signal routing tasks.

This library requires the following entries in the physical and application layer INI file.

physical layer INI file:

```
[device->SftDMM]
Description       = "Self Test Digital Multimeter"
Type              = NI4060
ResourceDesc      = DAQ::2::INSTR
DriverPrefix      = niDMM
DriverDll         = nidmm_32.dll
DriverOption      = Simulate=1,DriverSetup=PXI-4060
PowerLineFrequency = 50
```

```
;
[device->SftRelayCard]
Description  = "Self Test Matrix Card"
Type         = PMA1
ResourceDesc = PXI2::1::0::INSTR
DriverPrefix = rspma
DriverDll    = rspma.dll
DriverOption = "Simulate=1,DriverSetup=MCR:FFFFFFF6 CRAuto:1 BusSel:0"
;
[device->SampleDcs]
Description  = "Imaginary Sample Device"
Type         = DCS_SAMPLE
ResourceDesc = PXI2::4::0::INSTR
;
; mantadory analog bus entry
[device->ABUS]
Description = "Analog Bus"
Type        = ab
;
; hard wired connections
[io_channel->system]
DCS_HI = SftRelayCard!P11
DCS_LO = SftRelayCard!P13
```

application layer INI file:

```
[ResourceManager]
;
; Global tracing flags
Trace          = 1
TraceToScreen  = 0
TraceTimeStamp = 1
TraceThreadID  = 0
TraceFile      = c:\temp\trace.txt
;
[bench->SFT]
Trace             = 1
Simulation        = 1
DigitalMultimeter = device->SftDMM
SwitchDevice      = device->SftRelayCard
;
[SftOptions]
SystemName          = TS7100
SFTFixture          = 1
ManualInterventions = 1
ReportFile          = c:\temp\sft_report.txt
ReportStyle         = 3
ReportAppend        = 0
SuppressDialog      = 0
StopOnFirstFailure  = 0
;
; Self test parts
; Format: "PartX" = PartName, BenchName, SelectFlag
; The PartName must be unique for the whole section!
[SftParts]
Part1 = TSVP,   TSVP,   1
Part2 = SAMPLE, SAMPLE, 1
;
; TSVP self test bench
[bench->TSVP]
Trace = 1
Simulation = 1
;
; Self test bench for the sample library
[bench->SAMPLE]
Trace          = 1
```

```
Simulation      = 1
AnalogBus       = device->ABUS
SwitchDevice1   = device->SftRelayCard
DCS             = device->SampleDcs
AppChannelTable = io_channel->Sample
;
; channel table for the self test sample library
[io_channel->Sample]
ABa1 = ABUS!ABa1
ABa2 = ABUS!ABa2
```

The keys ”Trace” and ”Simulation” are all set to ”1” to allow debugging without any hardware.

11.5.3.4 Self test library structure

The exported ”self test function” has the following structure:

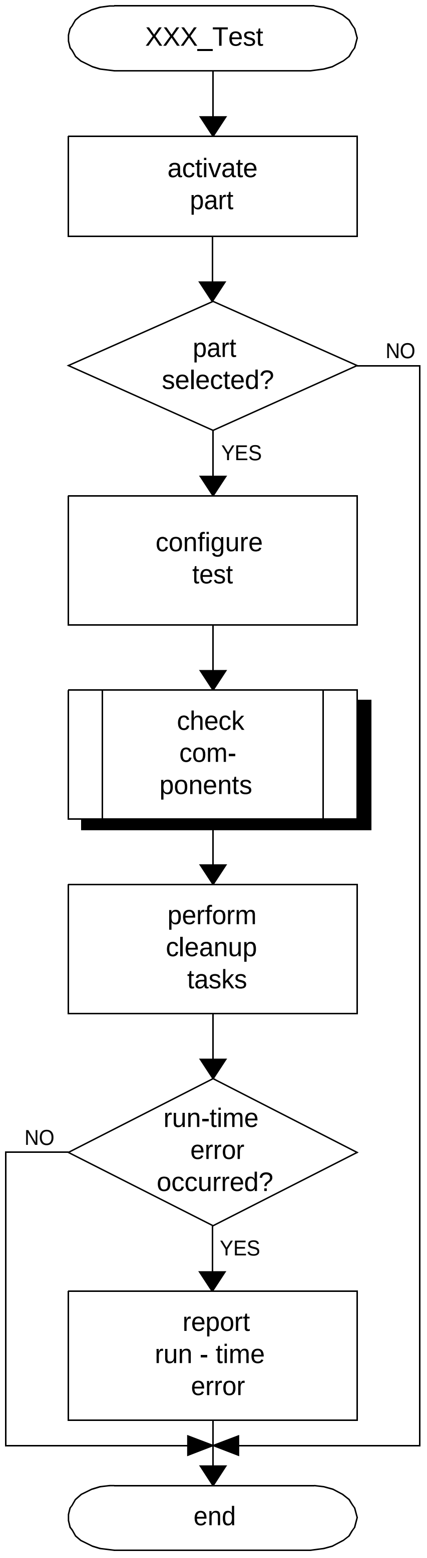


Figure 11-7 Exported self test function

The subroutine ”check components” has the following structure:

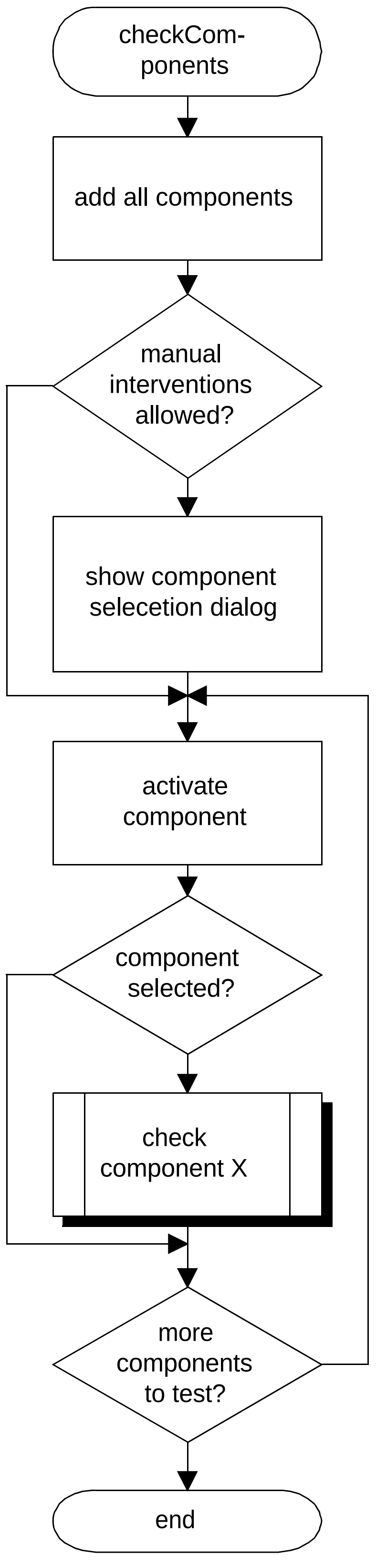


Figure 11-8 Check components function

The subroutine ”check component X” has the following structure:

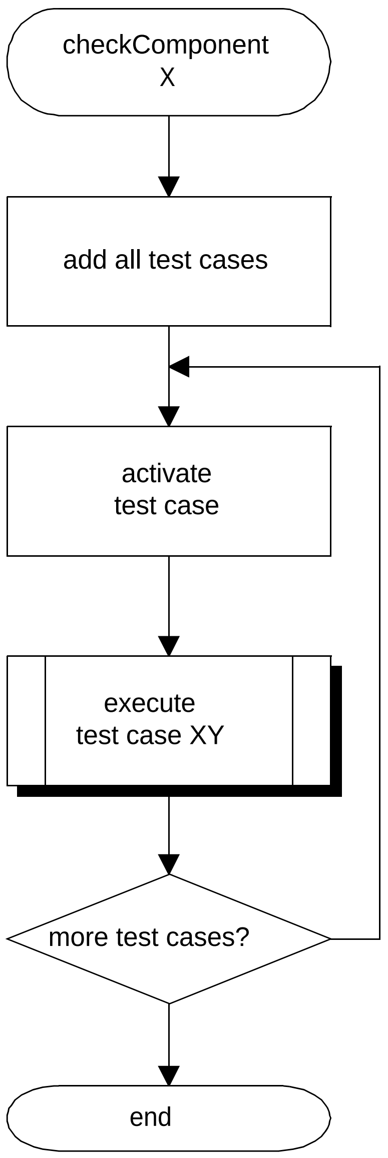


Figure 11-9 Check single component

The subroutine ”execute test case X” has the following structure:

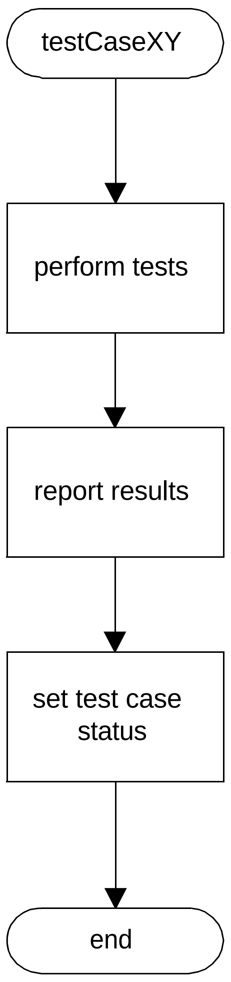


Figure 11-10 Test Case Function

11.5.3.4.1 Self Test Function

This function is called from the self test sequence. Be sure that the SFT_Setup function was called before. It performs the following tasks:

- Selects the given part
- Checks whether the given part is selected
- Gets the bench name related to the given part
- Allocates the bench resource
- Checks for tracing flag
- Checks the resource type
- Checks for simulation flag
- Opens the Switch Manager

- Gets the option flags set for the whole self test sequence
- Calls the function to perform the tests
- Closes the Switch Manager
- Frees the bench resource
- If an severe error occured it writes a message to the self test report

See function “SFTCSAMPLE_Test” in the sample project.

11.5.3.4.2 Components creation and dispatch function

This function is called from the exported self test function if the part is selected to be tested. It performs the following tasks:

- Creates all components for that part
- Shows the component select dialog if allowed
- Checks whether a component is selected to be tested
- Calls the appropriate component test function.

See function “checkComponents” in the sample project.

11.5.3.4.3 Component test function

If a component is selected to be tested, this function will be called by the component dispatch function. It executes all the test cases for the component. It creates all the test cases in the self test support library. The test cases are selected and the tests are done in this routine or by calling a test case function. It is the responsibility of this function to check some preconditions before calling a test case routine. If a test case can’t be executed a comment is added to the report to inform the user for the reason.

See functions “checkComponentDcs” and “checkComponentAux” in the sample project.

11.5.3.4.4 Test case function

It is called from the test case dispatcher (the component test function).

The test case is already created and selected by the dispatch routine. The preconditions are already chekked. It normally performs some measurements (e.g. with the SFT_Dmm_ functions) and reports the results with help of the self test support library. Finally it sets the test case status.

See functions “testDcsVoltage”, “testDcsDeviceSft”, “testAuxError-

Item" or "testAuxTableItem" in the sample project.

11.5.4 Resource Description

See chapter 9.5.3: Resource description

11.5.5 Miscellaneous

11.5.5.1 Error Handling

Error handling in a self test library is a little bit different from the handling in high - level libraries. The programmer must decide how to report errors from the underlying drivers or libraries.

Only unexpected or severe errors are handled in the way described in chapter 9.5.4.1: Error handling. Such errors will cause TestStand to pop up a "Run – Time Error" dialog. Additionally the programmer of the self test library is responsible that this error text is written to the self test report by using the appropriate function in the self test support library.

Error messages from device drivers or the resource manager that come from a faulty configuration in the INI files should be written to the self test report in an appropriate test case. Such an error should cause the test case to fail.

NOTE:

All functions of the self test support library report warnings (ErrorOccured is FALSE and ErrorCode is greater than zero) if the user aborts the self test or a test case failed and the option "Stop on first failure" is active. This warnings must lead to a immediate normal termination of the self test function. See chapter 11.5.5.6 for details.

11.5.5.2 Locking

See chapter 9.5.4.2: Locking for details.

11.5.5.3 Tracing

See chapter 9.5.4.3: Tracing for details.

11.5.5.4 Simulation

The implementation of simulation is not mandatory for a self test library. But it is recommended to support it. The reasons for running a self test library in simulation mode are:

- The self test sequence can be programmed and tested without hardware
- Presentation of the self test sequence without hardware
- Generation of a full report with all test cases passed

Simulation is done at a very high level in the library. During simulation mode, the high-level library must not call any device driver function or any other function requiring more than the standard PC hardware resources. The library functions should return some “typical” measured values to generate a “Pass” condition.

Simulation is enabled by the “Simulation” keyword in the appropriate bench section.

In the Setup section of the self test routine, the presence of the “Simulation” keyword is checked and the value of the simulation flag is stored.

11.5.5.5 Version Handling

The version number of a library is handled in two separate places.

- First, there is a **version string** which is written to the self test report.
- Second, there is a built-in **version number** in each DLL, which can be set during the DLL build. See chapter 9.5.4.6: Version handling” for details.

11.5.5.6 Self test abort

While the self test sequence is running a dialog with a “Abort” button is shown. When this button is activated by the user the event is stored in the self test support library. Every subsequent call will then return a warning. Another warning will be returned if a test case fails an the option “stop on first failure” is active. This warnings must lead to a immediate normal termination of the self test function. When all self test functions of the sequence will act in the same way the self test will terminate immediately.

11.5.6 CVI project structure

See chapter 9.5.5: CVI project structure for details.

11.6 SFT Sample Project

The self test sample project shows how a library interacts with the resource manager, the self test support library and the device driver functions.

The sample project is available in the following path

```
...\Gtsl\Develop\Libraries\Sftcsample
```

12 Path Characterization

12.1 General

Inaccuracies will occur when performing measurements that are contingent upon the physical characteristics of the measuring and stimuli paths (connections).

In order for the UUT (Unit Under Test) to have a defined signal output applied to it [P_{IN_UUT}], the stimuli device (signal generator) must issue a value greater than [P_{OUT}]. This ensures that losses [ΔP] are compensated at the connections.

$P_{OUT} = P_{IN\_UUT} + \Delta P$

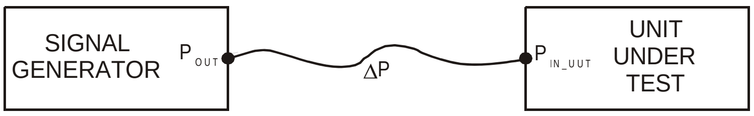


Figure 12-1 Signal Stimulation

The losses [ΔP] at the connections mean that when measuring signals, the value determined by the test unit (power indicator) [P_{IN}] is less than the actual value on the UUT [P_{OUT_UUT}]. Therefore, values determined by the test units must be revised upwards.

$P_{OUT\_UUT} = P_{IN} + \Delta P$

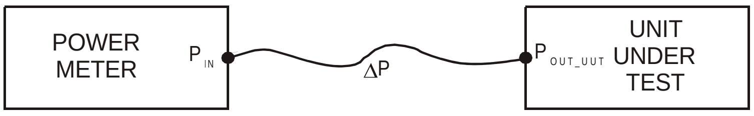


Figure 12-2 Signal Measurement

When determining precise test results these inaccuracies or losses [ΔP] must be taken into consideration. The correction values for the individual measurements are filed in the correction tables for the various frequencies. These correction tables are filed in Calibration Files for the various measurements and stimuli paths (see chapter 7.4.2.4 and chapter 12.2). The content of the Calibration Files is then considered by the individual functions in the High Level Libraries.

The individual measurement and stimuli paths (connections) can be made up of individual cables as well as a combination of different cables and transmission links.

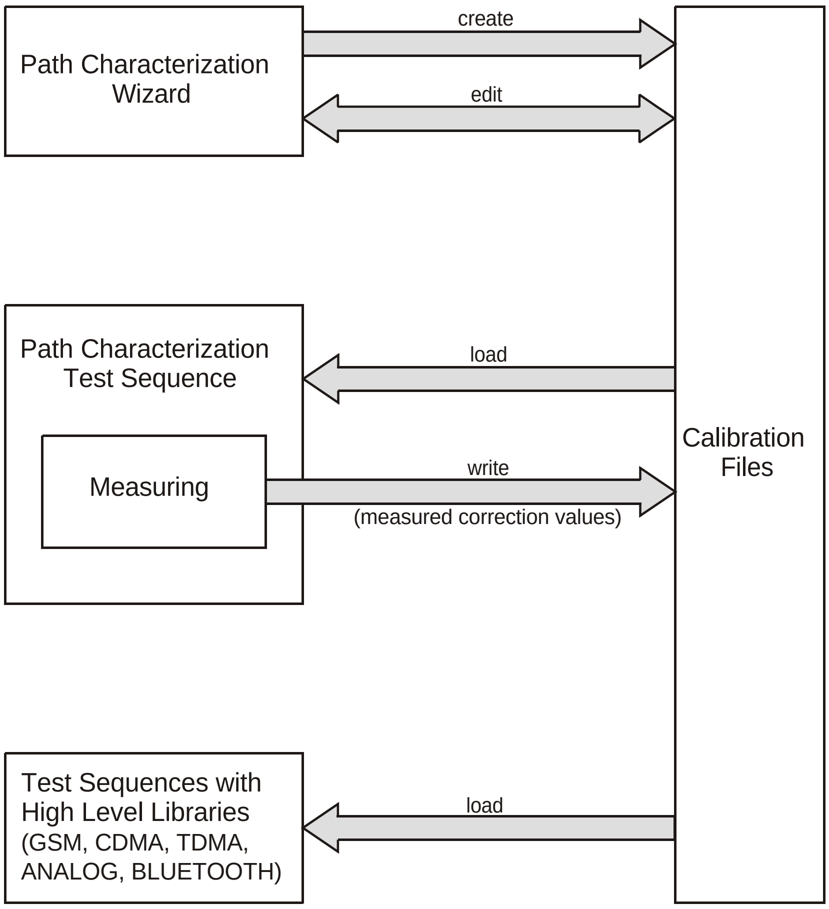

Path Characterization Wizard
create
edit
Path Characterization Test Sequence
Measuring
load
write
(measured correction values)
Calibration Files
Test Sequences with High Level Libraries (GSM, CDMA, TDMA, ANALOG, BLUETOOTH)
load

Figure 12-3 Generating and Using Calibration Files

Calibration Files are generated and used in several steps (see Figure 12-3):

1. The basic generation of the Calibration Files is done using the Path Characterization Wizard (see chapter 12.3). The Path Characterization Wizard can also be used to edit existing Calibration Files.
2. The determination of correction values and the entry of these correction values into the generated Calibration Files or correction tables is done using special Path Characterization test sequences (see chapter 12.4). These test sequences use functions from the Path Characterization Library.

   When running the Path Characterization test sequence exact details are given as to how the individual measurement is to be performed for determining the correction values.

3. Using Calibration Files in the High Level Libraries.

   The reference to the individual Calibration Files is defined in Section [bench->...] of the corresponding Test Library in the APPLICATION.INI file. See also the help file „pathchar.hlp“ of the Path Characterization Library and the descriptions of the „Calibration Entries“ in the according Test Libraries in Sections 7.x.x.3 (Entries in APPLICATION.INI).

12.2 Calibration File

The Calibration Files are, as everywhere in the R&S GTSL, structured similar to a Windows INI file.

The `...\GTSL\Configuration` directory contains a Calibration File as an example (file name: `Demo_CH1_GSM.CAL`).

NOTE:

The content of the various Calibration File sections are described in Chapter 7, Section “7.4.2.4: Entries in CALIBRATION.CAL”.

12.3 Path Characterization Wizard

12.3.1 General

NOTE:

Further information is available in the online help for the Path Characterization Wizard. The online help is called up using the F1 soft key or the <Help><Documentation> menu item.

The online help contains description on the individual menus, buttons and windows in the Path Characterization Wizard. At this point, only the operational steps are described, which are required for generation of a new Calibration File or for editing an existing Calibration File.

12.3.2 New Calibration File Generation

12.3.2.1 Create Calibration File

Perform the following operations to create a new Calibration File.

pcwizard

1. Run the Path Characterization Wizard over **Start -> Programs -> GTSL -> Tools -> Path Characterization Wizard**. The Path Characterization Wizard can also be run from a shortcut icon placed on the desktop.

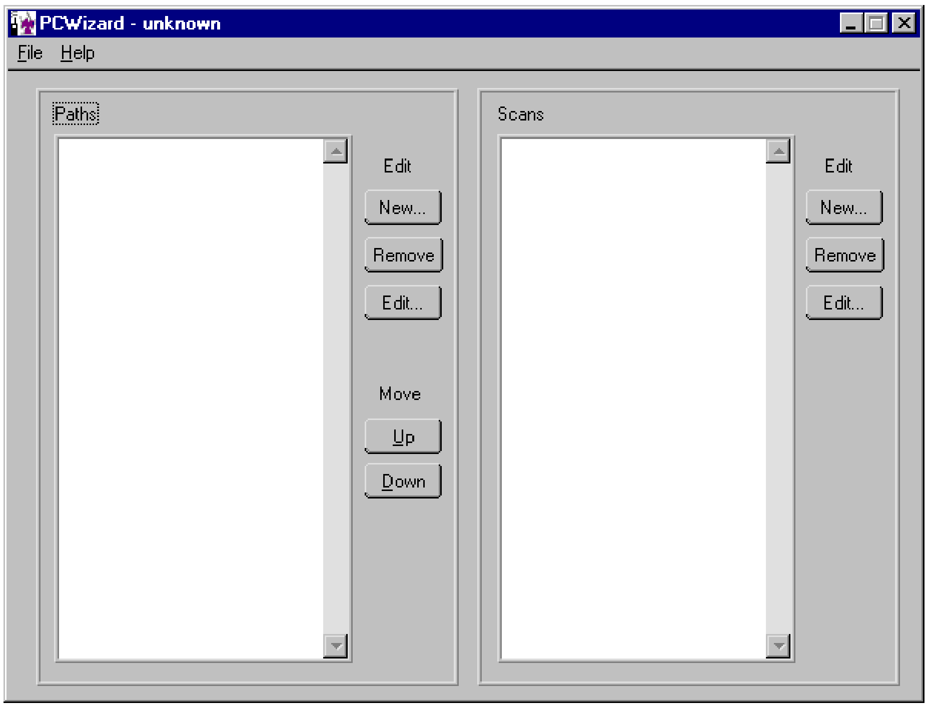


Figure 12-4 Main Window Path Characterization Wizard

2. Using the **New...** button create a new path(s) in the Path Manipulation Section and edit it/them (see chapter 12.3.2.2).
3. Using the **New...** button create a new scan or scans in the Scan Manipulation Section and edit it/them (see chapter 12.3.2.3 and 12.3.2.4).
4. Allocate the necessary scans to the newly generated path or paths (see chapter 12.3.2.5).
5. Using <File><Save> or <File><Save as...> the newly generated Calibration File can be saved.
6. Close the Path Characterization Wizard using <File><Exit>.
7. Running Path Characterization test sequences. The correction

value determined when performing the test sequences are entered into the newly created Calibration File. See also the help file in the „pathchar.hlp“ file of the Path Characterization Library.

12.3.2.2 Editing / Creating Paths

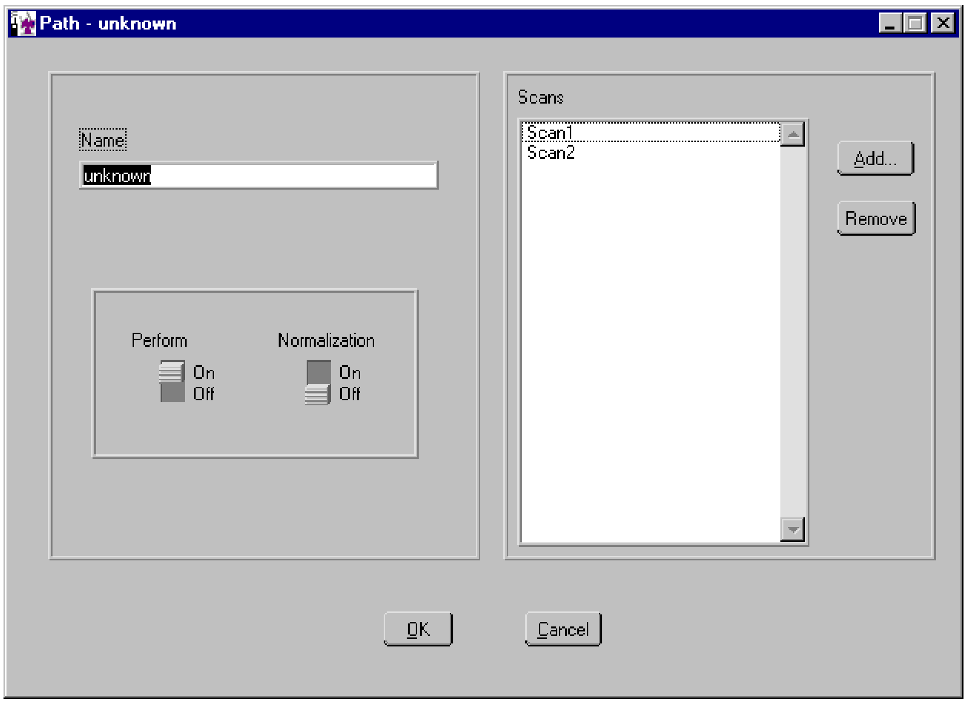


Figure 12-5 Path Edit Window

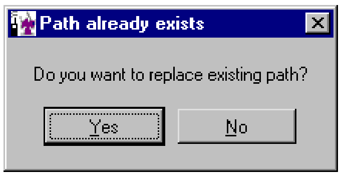
If either the **Edit...** button or the **New...** button is selected in the Path Manipulation Section (see Figure 12-4) the Path Edit Window (Figure 12-5) will appear. The left side section provides possibilities to change the name as well as the Perform Flag and Normalization Flag.

NOTE:

In the calibration file „PathInfo->“ is prefixed to the name of the path. This will not be used for editing. Therefore feeding „PathInfo->unknown“ in the Name-field will result in a section „PathInfo->PathInfo->unknown“.

In the other section (on the right side) you can remove selected scans from the List of scans to be performed (Scan Reference List) or open a window (Scan Reference Window) in which you edit the list. The Scan Reference Window (see Figure 12-18) will be called when using the **Add...** button or pressing <ALT><A>.

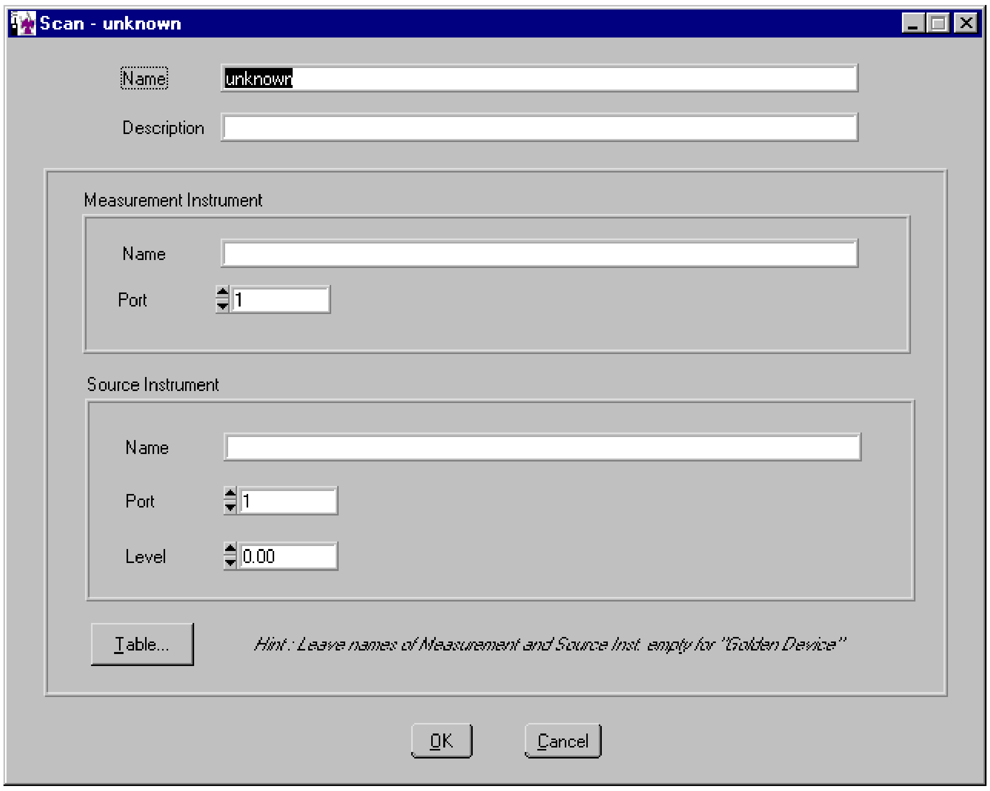
Below the two sections are the accept and decline buttons. When choosing the button **Cancel** made changes are discarded, whereas choosing the button **OK** will accept changes. When accepting changes a new path will be inserted at the end of the list and an edited path will stay at its position. As the paths in the calibration file only differ by name, there cannot be two paths having the same name. For this reason if the name specified for the current path matches to another path in the list, Figure 12-6 will be displayed asking you to replace the existing one. When editing a path and confirming replacement of an existing one, the number of paths in the list will be one less than before, because the replaced path doesn't exist any more and there also is no copy of the replacing path having the old name.


Path already exists

Do you want to replace existing path?

Figure 12-6 Replace Path Dialog

12.3.2.3 Editing / Creating Scans


Scan - unknown

Name: unknown
Description:

Measurement Instrument
Name:
Port: 1

Source Instrument
Name:
Port: 1
Level: 0.00

Hint : Leave names of Measurement and Source Inst. empty for "Golden Device"

Figure 12-7 Scan Edit Window

If either the **Edit...** button or the **New...** button is selected in the Scan Manipulation Section (see Figure 12-4) the Scan Edit Window (Figure 12-7) will appear.

There three input sections can be differentiated:

General Description:

Name	The name of the scan (as well as the section name in calibration file). **Note:** The title bar also shows the name of the scan.
Description	A short description for the scan.

Measurement Instrument:

Name	Name of the measurement instrument. See Scan Section [ScanName] in section “7.4.2.4: Entries in CALI-BRATION.CAL”
Port	Port of the measurement instrument see. See Scan Section [ScanName] in section “7.4.2.4: Entries in CALI-BRATION.CAL”

Source Instrument:

Name	The name of the source instrument. See Scan Section [ScanName] in section “7.4.2.4: Entries in CALIBRATION.CAL”
Port	Port of the source instrument. See Scan Section [ScanName] in section “7.4.2.4: Entries in CALIBRATION.CAL”
Level	Level of source instrument. See Scan Section [ScanName] in section “7.4.2.4: Entries in CALIBRATION.CAL”

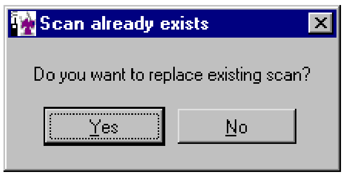
Table...	Open the Table Edit Window (see Figure 12-9)
OK	Accept changes.
Cancel	Decline changes.

NOTE:

If text boxes of measurement instrument and source instrument name are left empty the respective section in the calibration file will be deleted representing a „Golden Device“.

As scans involve a table of frequencies at which path characterization is to be performed, the **Table...** button in the left bottom of the window will guide you to the Table Edit Window, where you can edit the table of the current scan. See chapter 12.3.2.4: Editing the Table of a Scan.

Similar to chapter 12.3.2.2: Editing / Creating Paths, there are two buttons beneath for accept and decline. When choosing the **Cancel** button made changes are discarded, whereas choosing the **OK** button will accept changes. When accepting changes a new scan will be inserted in alphanumeric order and an edited scan will stay at its position respectively. As the scans in the calibration file like paths only differ by name, there cannot be two scans having the same name. For this reason if the name specified for the current scan matches to another one in the list, the window as shown in Figure 12-8 will be displayed asking you to replace the existing one. When editing a scan and confirming replacement of an existing one, the number of scans in the list will be one less than before, because the replaced scan doesn't exist any more and there also is no copy of the replacing scan having the old name.

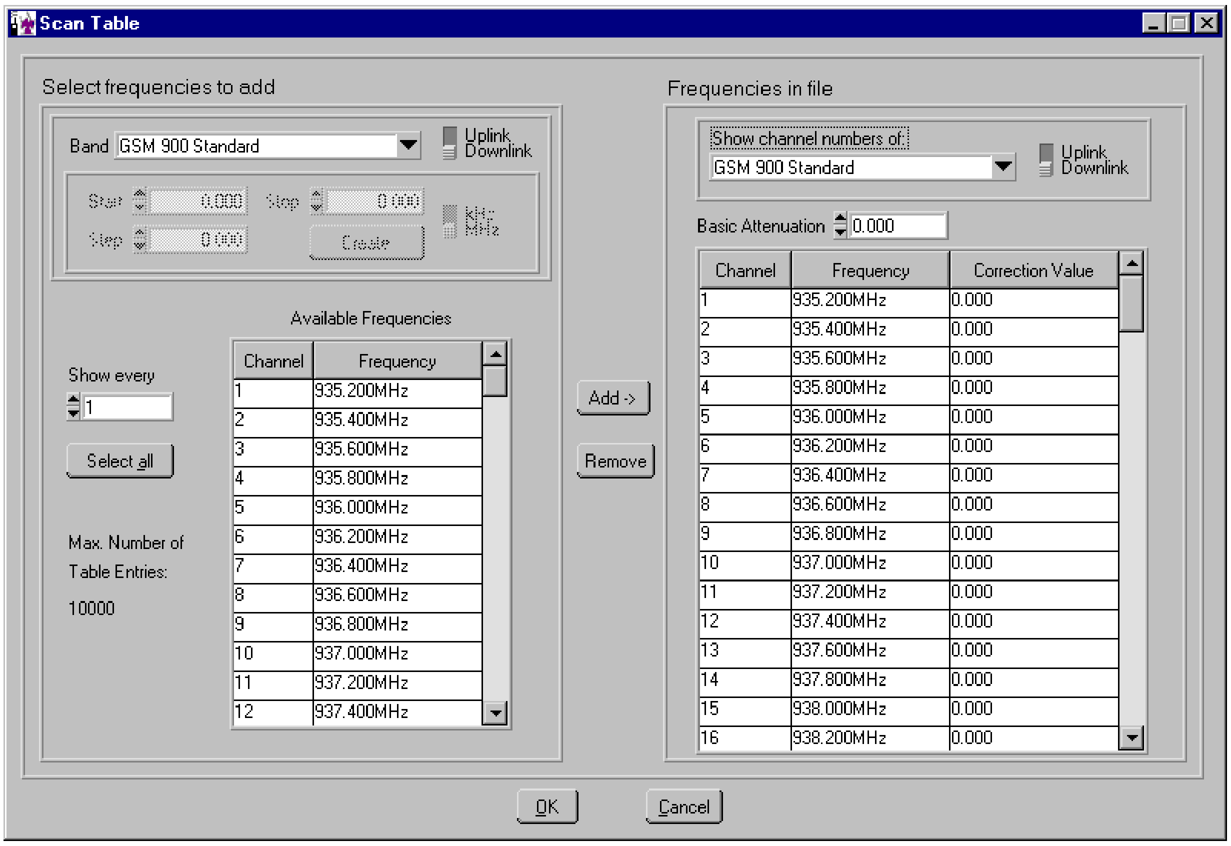

Scan already exists

Do you want to replace existing scan?

Yes | No

Figure 12-8 Replace Scan Dialog

12.3.2.4 Editing the Table of a Scan


Scan Table

Select frequencies to add

Band: GSM 900 Standard — Uplink / Downlink

Start 0.000 — Stop 0.000 — Step 0.000 — kHz / MHz — Create

Show every: 1

Select all

Max. Number of Table Entries: 10000

Available Frequencies

Channel	Frequency
1	935.200MHz
2	935.400MHz
3	935.600MHz
4	935.800MHz
5	936.000MHz
6	936.200MHz
7	936.400MHz
8	936.600MHz
9	936.800MHz
10	937.000MHz
11	937.200MHz
12	937.400MHz

Add -> | Remove

Frequencies in file

Show channel numbers of: GSM 900 Standard — Uplink / Downlink

Basic Attenuation: 0.000

Channel	Frequency	Correction Value
1	935.200MHz	0.000
2	935.400MHz	0.000
3	935.600MHz	0.000
4	935.800MHz	0.000
5	936.000MHz	0.000
6	936.200MHz	0.000
7	936.400MHz	0.000
8	936.600MHz	0.000
9	936.800MHz	0.000
10	937.000MHz	0.000
11	937.200MHz	0.000
12	937.400MHz	0.000
13	937.600MHz	0.000
14	937.800MHz	0.000
15	938.000MHz	0.000
16	938.200MHz	0.000

OK | Cancel

Figure 12-9 Table Edit Window

This window is shown when the **Table...** button was selected in the Scan Edit Window (see Figure 12-7). It helps you editing the frequency table of the calibration file by providing two main sections:

- The left section is for Selecting Frequencies/Channels to add (see chapter 12.3.2.4.1 ff.).
- The right section is for Editing the Table and also shows channel

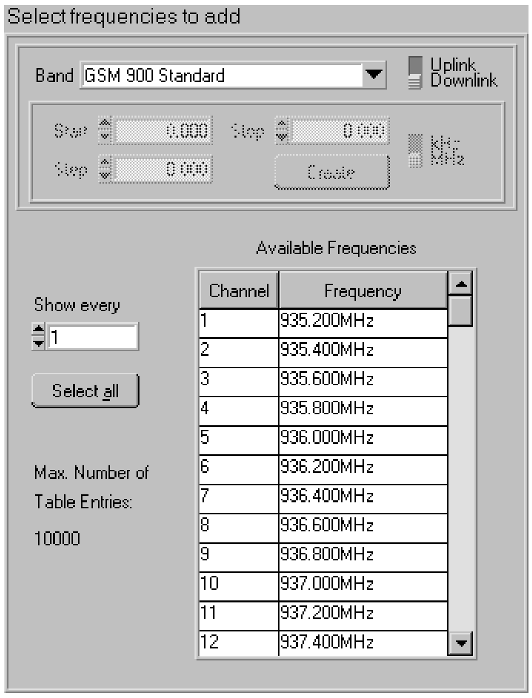
information (see chapter 12.3.2.4.4).

Using the **Add ->** button you can add the items selected in the table of available frequencies to the list of frequencies in the calibration file (right table). You also can add a single one by double-clicking on it.

The Remove button can be used to remove selected frequencies from the list of frequencies in file and therefore takes effect on the right table.

At the bottom of the window the known buttons for accepting and declining changes are included.

12.3.2.4.1 Selecting Frequencies / Channels to add



Band: GSM 900 Standard

Start 0.000 Stop 0.000

Step 0.000

Show every 1

Max. Number of Table Entries: 10000

Available Frequencies

Channel	Frequency
1	935.200MHz
2	935.400MHz
3	935.600MHz
4	935.800MHz
5	936.000MHz
6	936.200MHz
7	936.400MHz
8	936.600MHz
9	936.800MHz
10	937.000MHz
11	937.200MHz
12	937.400MHz

Figure 12-10 Selecting Frequencies/Channels to add Section

The section shown in Figure 12-10 is intended to help you creating the frequencies you want to be added to your scan for path characterization. The upper part is for creating frequencies whereas the other part is for selecting them.

12.3.2.4.2 Creating Frequencies

There are two ways of creating frequencies, both accessible with band selection. Please note that you can add both, channels and arbitrary frequencies to the right table. See also section 12.3.2.4.3: Selection options.

Adding Channels of selectable Bands to your Scan

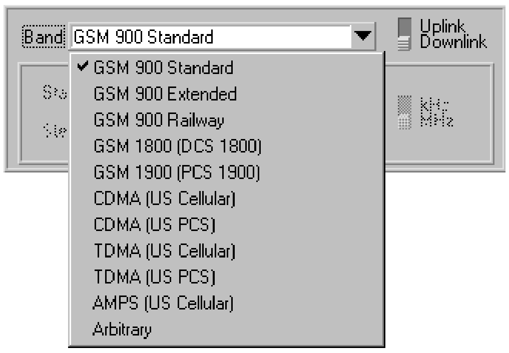


Figure 12-11 Scan Table Band Selection

Use the ring control in Figure 12-11 to choose the band with the appropriate channels.

„Arbitrary“ is not a specified band and will be explained later (Adding special frequencies). Therefore choosing for example GSM 900 Standard will look like in Figure 12-12. You also can use the toggle button on the right to switch between uplink and downlink channels.

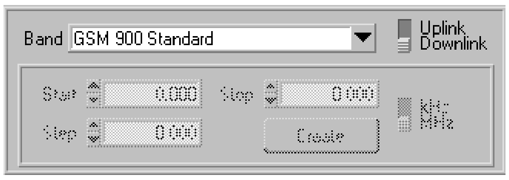


Figure 12-12 Scan Table - GSM 900 Standard Selection

Depending on the band selected the channels will be listed in ascending order (regarding frequency) in the table of available frequencies.

Adding special Frequencies

The other way of creating the table is selecting Arbitrary in the band selection ring and adding special frequencies to the table. In arbitrary mode the toggle button is grey, but some numeric inputs are now available. These inputs help you defining an interval with the frequencies you want.

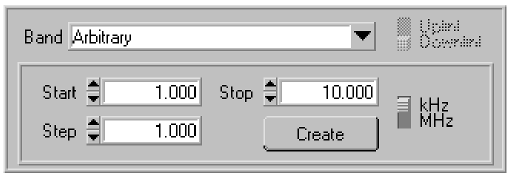


Figure 12-13 Scan Table - Arbitrary Selection

The inputs are:

Start	Determining the start frequency of the interval (always part of the list if not above stop frequency).
Stop	Determining the stop frequency of the interval.
Step	Determining the step width within the interval. If it is greater than the difference between start and stop frequency only the start frequency will be part of the list.

Using the **kHz - MHz** toggle button specifies the unit of your input. Click on the Create button to show the interval in the table of available frequencies.

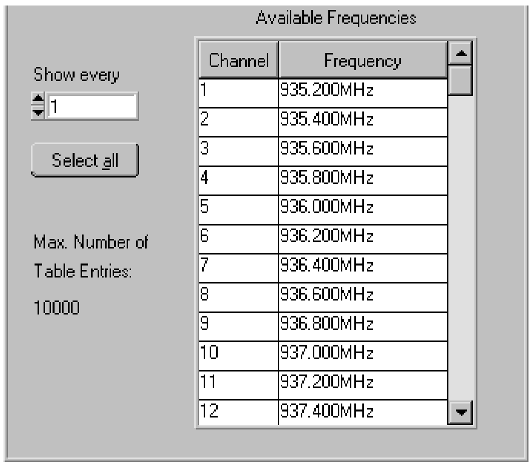
Examples (MHz selected):

Start = 10, Stop = 10, Step = 10:	Creates only {10 MHz} because stop frequency is smaller than next step (= 20 MHz = Start + Step).
Start = 10, Stop = 30, Step = 10:	Creates {10 MHz, 20 MHz, 30 MHz}.
Start = 20, Stop = 10, Step = 5:	Creates empty list because stop frequency is smaller than start frequency.

NOTE:

The number of table rows is limited to 10,000. Therefore only the first 10,000 steps of the defined interval will be inserted.

12.3.2.4.3 Selection options



Channel	Frequency
1	935.200MHz
2	935.400MHz
3	935.600MHz
4	935.800MHz
5	936.000MHz
6	936.200MHz
7	936.400MHz
8	936.600MHz
9	936.800MHz
10	937.000MHz
11	937.200MHz
12	937.400MHz

Figure 12-14 Scan Table - Available Frequencies

After creation of the table the frequencies now can be added to scan. For this purpose you need to select the frequencies within the table. A single frequency can be selected by just clicking on the according table cell. For a range of frequencies use the <SHIFT> key and either arrow keys or mouse button. Selecting all frequencies in the list is possible by using <ALT><A>, the **Select all** button or simply clicking on the headline of a column.

Another way of manipulating the list is displaying only every n-th channel. Use the **Show every** input to specify n.

As (for example) GSM 900 Extended starts with channel 975 there possibly is the need for a search function within the table, so that you can find channel 1. When right clicking on the table the following popup (see Figure 12-15) appears:

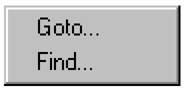


Figure 12-15 Table Popup

Goto... is used to jump to a cell, whereas **Find...** will bring up the following dialog:

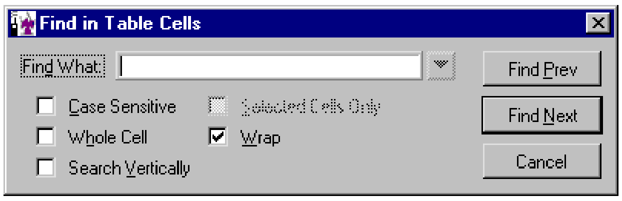


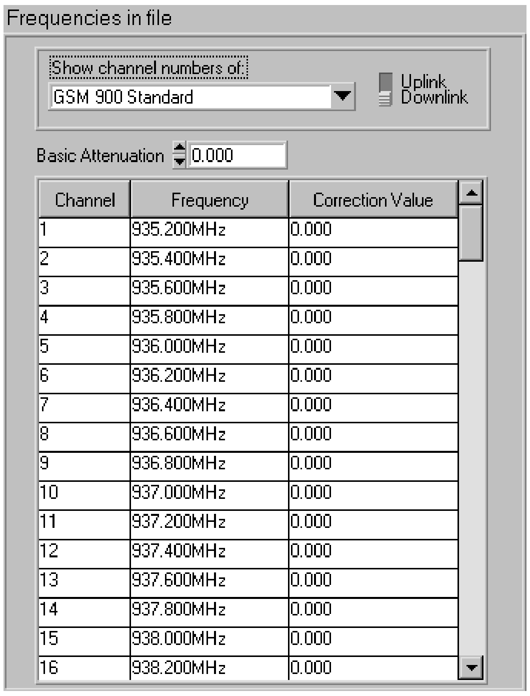
Figure 12-16 Table Find Dialog

It is recommended to check Whole Cell to search for a channel number, because a 1 is also part of channel 981.

Adding frequencies

As mentioned earlier this chapter the **Add...** button is used to add your selection in the table of available frequencies to the table of frequencies of the scan. Please note that not only single selections can be added. Creating frequencies with the left side of the window you are able by creating, selecting, adding to built up step by step the table you want.

12.3.2.4.4 Editing the Table



Show channel numbers of: GSM 900 Standard

Basic Attenuation 0.000

Channel	Frequency	Correction Value
1	935.200MHz	0.000
2	935.400MHz	0.000
3	935.600MHz	0.000
4	935.800MHz	0.000
5	936.000MHz	0.000
6	936.200MHz	0.000
7	936.400MHz	0.000
8	936.600MHz	0.000
9	936.800MHz	0.000
10	937.000MHz	0.000
11	937.200MHz	0.000
12	937.400MHz	0.000
13	937.600MHz	0.000
14	937.800MHz	0.000
15	938.000MHz	0.000
16	938.200MHz	0.000

Figure 12-17 Scan Table - Frequency List

In the section displayed in Figure 12-17 the table represents the frequencies in the calibration file used for path characterization.

As the correction value is calculated from the basic attenuation plus the frequency depending level changing the value of the Basic Attenuation input will also take effect on the values of the table. Therefore changing it by +1 will result in decreasing all levels by 1.

To edit the levels directly put the cursor on the cell you want the value to be changed and either press **F2** or double-click on it. Changing the value can be cancelled by pressing **ESC** in the edit mode. Any other action like pressing **ENTER** or clicking somewhere else will result in confirming the change.

At the top of the section a ring control and a toggle button can be found. These controls are intended to show the channel numbers corresponding to the frequencies of the table. Using the ring specifies the band while the toggle button distinguishes between uplink and downlink. If there is no channel number having the same frequency a dash is added instead of a number.

12.3.2.5 Editing Scan References of a Path

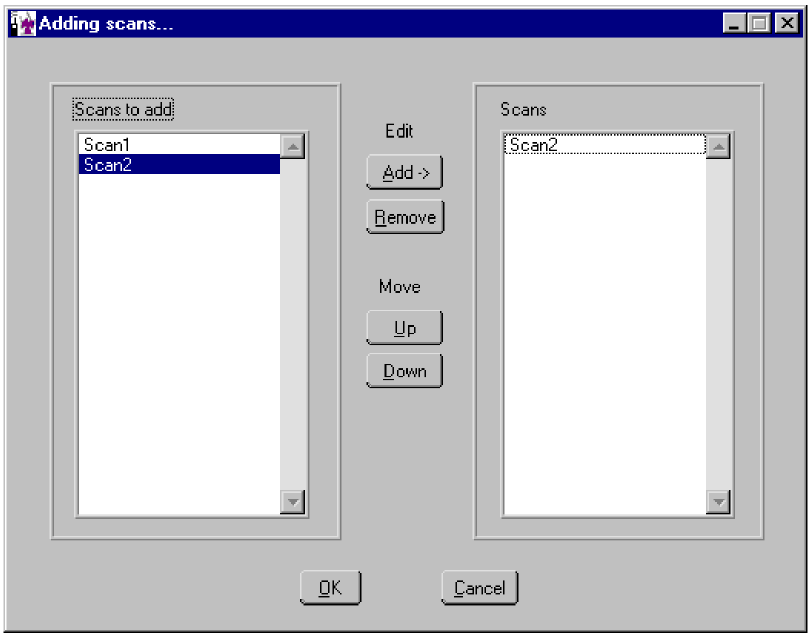


Figure 12-18 Scan Reference Window

The Scan Reference Window in Figure 12-18 will appear when selecting the **Add...** button in the Path Edit Window (see Figure 12-5) and helps you to make changes to the Scan Reference List of the current path. On the left side is a box indicating all scans which are defined yet.

The listbox on the right side is the Scan Reference List, which shows the scans to be done for path characterization. Between the two boxes are four buttons, all of them taking effect on the right box. The other two buttons are for accepting or not accepting changes. All buttons can also be accessed by pressing <ALT> in conjunction with the underlined letter.

Add ->	Adds a Scan Reference (name of scan) to the end of the list. Also available by double-clicking on scan in left box.
Remove	Removes selected scan from Scan Reference List.
Up	Moves selected Scan Reference towards beginning of list.
Down	Moves selected Scan Reference towards end of list.
OK	Accept changes.
Cancel	Decline changes.

12.3.3 Editing an Existing Calibration File

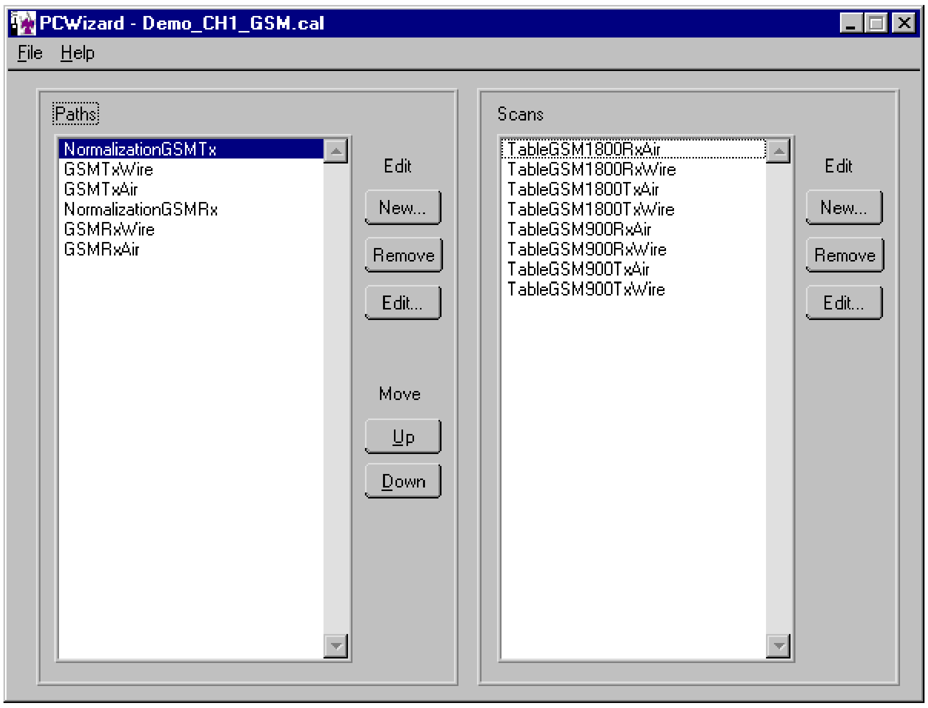
Perform the following operations to edit an existing Calibration File:

pcwizard

1. Run the Path Characterization Wizard over **Start -> Programs -> GTSL -> Path Characterization Wizard**. The Path Characterization Wizard can also be run from a shortcut icon placed on the desktop.
2. Open the Calibration File using <File><Open>.

   The opened Calibration File is displayed in the main Path Characterization Wizard window (see Figure 12-19).

   The Demo Calibration Files created and included in the delivery specification by ROHDE & SCHWARZ are filed in the `...\GTSL\Configuration` directory.



NormalizationGSMTx
GSMTxWire
GSMTxAir
NormalizationGSMRx
GSMRxWire
GSMRxAir

TableGSM1800RxAir
TableGSM1800RxWire
TableGSM1800TxAir
TableGSM1800TxWire
TableGSM900RxAir
TableGSM900RxWire
TableGSM900TxAir
TableGSM900TxWire

Figure 12-19 Main Window Path Characterization Wizard with demo file

3. Performing changes (see chapter 12.3.2.2 to 12.3.2.5).
4. Using <File><Save> or <File><Save as...> save the Calibration File.
5. Close the Path Characterization Wizard using <File><Exit>.

12.4 Determining Correction Values

12.4.1 General

The determination of correction values and the entry of these correction values into the generated Calibration Files is done over the Path Characterization test sequences.

The `...\GTSL\Sequences` directory contains an example test sequence (File name: `Demo_CH1_PathCharacterization.seq`).

NOTE:

Using this Path Characterization example test sequence serves to describe the principle process when determining the correction values.

This example test sequence can be used as a template for creating your own Path Characterization test sequences.

The `...\GTSL\Configuration` directory contains examples of Calibration Files and Configuration Files. The example files can also be used as templates.

The following units are supported for the purpose of determining correction values or performing measurements:

- Source Instrumentation
  - Radio Communication Tester CMU
  - Radio Communication Tester CMD55
  - Vector Signal Generator SMIQ
  - Signal Generator SME
- Measurment Instrumentation
  - Radio Communication Tester CMU
  - Radio Communication Tester CMD55
  - Power Meter NRVS
  - Dual-Channel Power Meter NRVD

12.4.2 Preparing Calibration Files

Before starting the Path Characterization test sequence the Path Characterization Wizard must be used to generate or prepare a Calibration File (see chapter 12.3). An example of a typical Calibration File is listed below.

Example: Calibration File for Wire Coupling	Example: Calibration File for Air Coupling
[PathInfo->NormalizationGSMTx]	[PathInfo->NormalizationGSMTx]
Perform = **1**	Perform = **1**
Normalization = **1**	Normalization = **1**
Scan1 = TableGSM900TxWire	Scan1 = TableGSM900TxWire
Scan2 = TableGSM1800TxWire	Scan2 = TableGSM1800TxWire
[PathInfo->GSMTxWire]	[PathInfo->GSMTxWire]
Perform = **1**	Perform = **0**
Scan1 = TableGSM900TxWire	Scan1 = TableGSM900TxWire
Scan2 = TableGSM1800TxWire	Scan2 = TableGSM1800TxWire
[PathInfo->GSMTxAir]	[PathInfo->GSMTxAir]
Perform = **0**	Perform = **1**
Scan1 = TableGSM900TxAir	Scan1 = TableGSM900TxAir
Scan2 = TableGSM1800TxAir	Scan2 = TableGSM1800TxAir
[PathInfo->NormalizationGSMRx]	[PathInfo->NormalizationGSMRx]
Perform = **1**	Perform = **1**
Normalization = **1**	Normalization = **1**
Scan1 = TableGSM900RxWire	Scan1 = TableGSM900RxWire
Scan2 = TableGSM1800RxWire	Scan2 = TableGSM1800RxWire
[PathInfo->GSMRxWire]	[PathInfo->GSMRxWire]
Perform = **1**	Perform = **0**
Scan1 = TableGSM900RxWire	Scan1 = TableGSM900RxWire
Scan2 = TableGSM1800RxWire	Scan2 = TableGSM1800RxWire
[PathInfo->GSMRxAir]	[PathInfo->GSMRxAir]
Perform = **0**	Perform = **1**
Scan1 = TableGSM900RxAir	Scan1 = TableGSM900RxAir
Scan2 = TableGSM1800RxAir	Scan2 = TableGSM1800RxAir
[TableGSM900TxWire]	[TableGSM900TxAir]
Description = This table is valid for uplink	Description = This table is valid for uplink
GSM900 band wire coupled	GSM900 band wire coupled
SourceInst = device->**CMU_CH1**	SourceInst = device->**CMU_CH1**
SourceLevel = **10.0**	SourceLevel = **-10.0**
SourcePort = **3**	SourcePort = **3**
MeasurementInst = device->**NRVS**	MeasurementInst = device->**NRVS**
MeasurementPort = 4	MeasurementPort = 4
BasicAttenuation = 1.2	BasicAttenuation = 1.2
.	.
.	.
[TableGSM900RxWire]	[TableGSM900RxAir]
Description = This table is valid for downlink	Description = This table is valid for downlink
GSM900 band wire coupled	GSM900 band wire coupled
SourceInst = device->**CMU_CH1**	SourceInst = device->**CMU_CH1**
SourcePort = **3**	SourcePort = **3**
SourceLevel = **10.0**	SourceLevel = **-10.0**
MeasurementInst = device->**NRVS**	MeasurementInst = device->**NRVS**
MeasurementPort = 4	MeasurementPort = 4
BasicAttenuation = 1.1	BasicAttenuation = 1.1
.	.
.	.
[TableGSM1800TxWire]	[TableGSM1800TxAir]
Description = This table is valid for uplink	Description = This table is valid for uplink
GSM1800 band wire coupled	GSM1800 band wire coupled
SourceInst = device->**CMU_CH1**	SourceInst = device->**CMU_CH1**
SourcePort = **3**	SourcePort = **3**

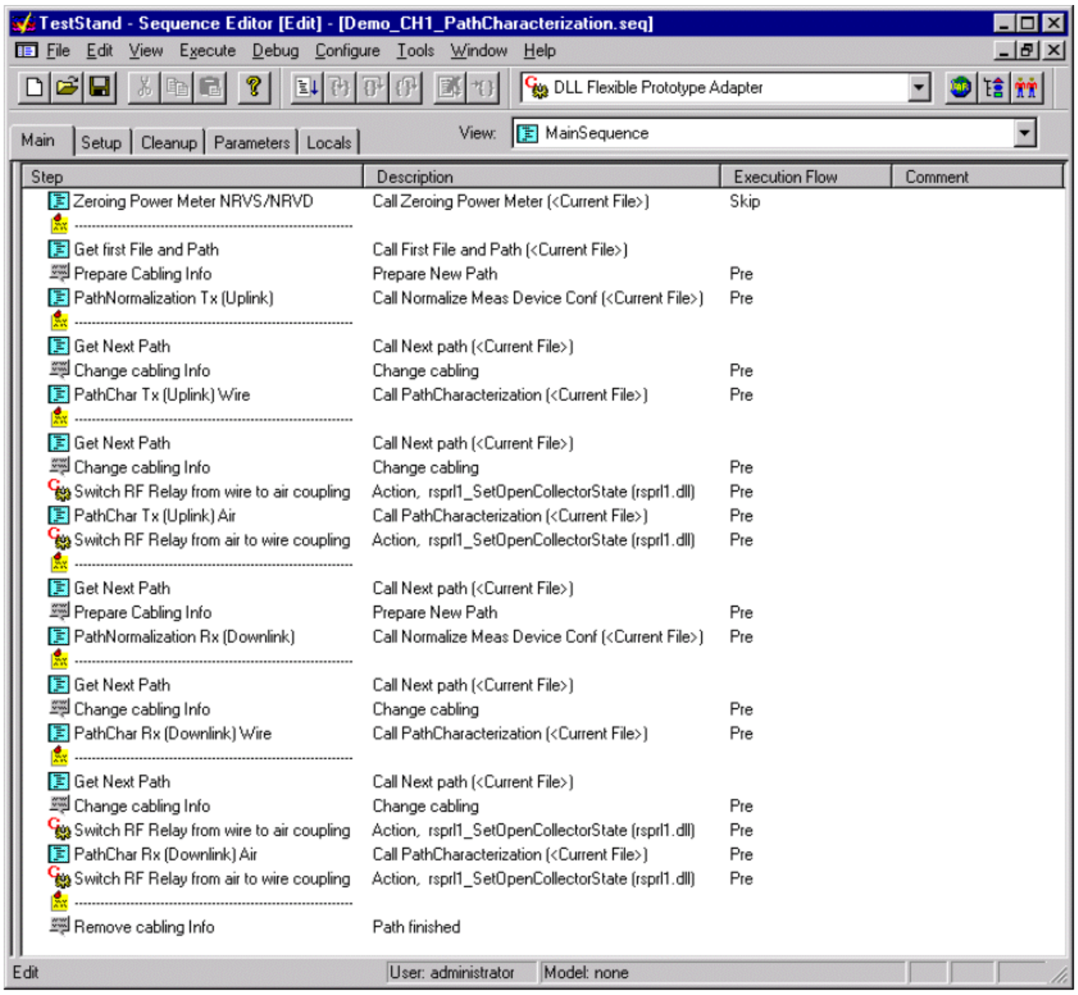
```
SourceLevel = 10.0
MeasurementInst = device->NRVS
MeasurementPort = 4
BasicAttenuation = 1.8
.
.

[TableGSM1800RxWire]
Description = This table is valid for downlink
              GSM1800 band wire coupled
SourceInst = device->CMU_CH1
SourcePort = 3
SourceLevel = 10.0
MeasurementInst = device->NRVS
MeasurementPort = 4
BasicAttenuation = 1.7
```

```
SourceLevel = -10.0
MeasurementInst = device->NRVS
MeasurementPort = 4
BasicAttenuation = 1.8
.
.

[TableGSM1800RxAir]
Description = This table is valid for downlink
              GSM1800 band wire coupled
SourceInst = device->CMU_CH1
SourcePort = 3
SourceLevel = -10.0
MeasurementInst = device->NRVS
MeasurementPort = 4
BasicAttenuation = 1.7
```

The highlighted entries are required for performing the Path Characterization test sequence. By setting the Perform Flag (0 = Measurements for the path are not executed, 1 = Measurements for the path are executed) the Path Characterization test sequence can be controlled. This way a test sequence and a Calibration File can be used for the measurements for a Wire Coupling and an Air Coupling. See also the typical measurement setups in Section 12.4.4.

12.4.3 Path Characterization Test Sequence Process.

The Path Characterization test sequence is called up and started as all other test sequences (see Chapter 6: Editing and running test sequences).


TestStand - Sequence Editor [Edit] - [Demo_CH1_PathCharacterization.seq]
File Edit View Execute Debug Configure Tools Window Help
DLL Flexible Prototype Adapter
Main Setup Cleanup Parameters Locals
View: MainSequence
Edit
User: administrator
Model: none

Step	Description	Execution Flow	Comment
Zeroing Power Meter NRVS/NRVD	Call Zeroing Power Meter (<Current File>)	Skip	

Get first File and Path	Call First File and Path (<Current File>)		
Prepare Cabling Info	Prepare New Path	Pre	
PathNormalization Tx (Uplink)	Call Normalize Meas Device Conf (<Current File>)	Pre	

Get Next Path	Call Next path (<Current File>)		
Change cabling Info	Change cabling	Pre	
PathChar Tx (Uplink) Wire	Call PathCharacterization (<Current File>)	Pre	

Get Next Path	Call Next path (<Current File>)		
Change cabling Info	Change cabling	Pre	
Switch RF Relay from wire to air coupling	Action, rsprl1_SetOpenCollectorState (rsprl1.dll)	Pre	
PathChar Tx (Uplink) Air	Call PathCharacterization (<Current File>)	Pre	
Switch RF Relay from air to wire coupling	Action, rsprl1_SetOpenCollectorState (rsprl1.dll)	Pre	

Get Next Path	Call Next path (<Current File>)		
Prepare Cabling Info	Prepare New Path	Pre	
PathNormalization Rx (Downlink)	Call Normalize Meas Device Conf (<Current File>)	Pre	

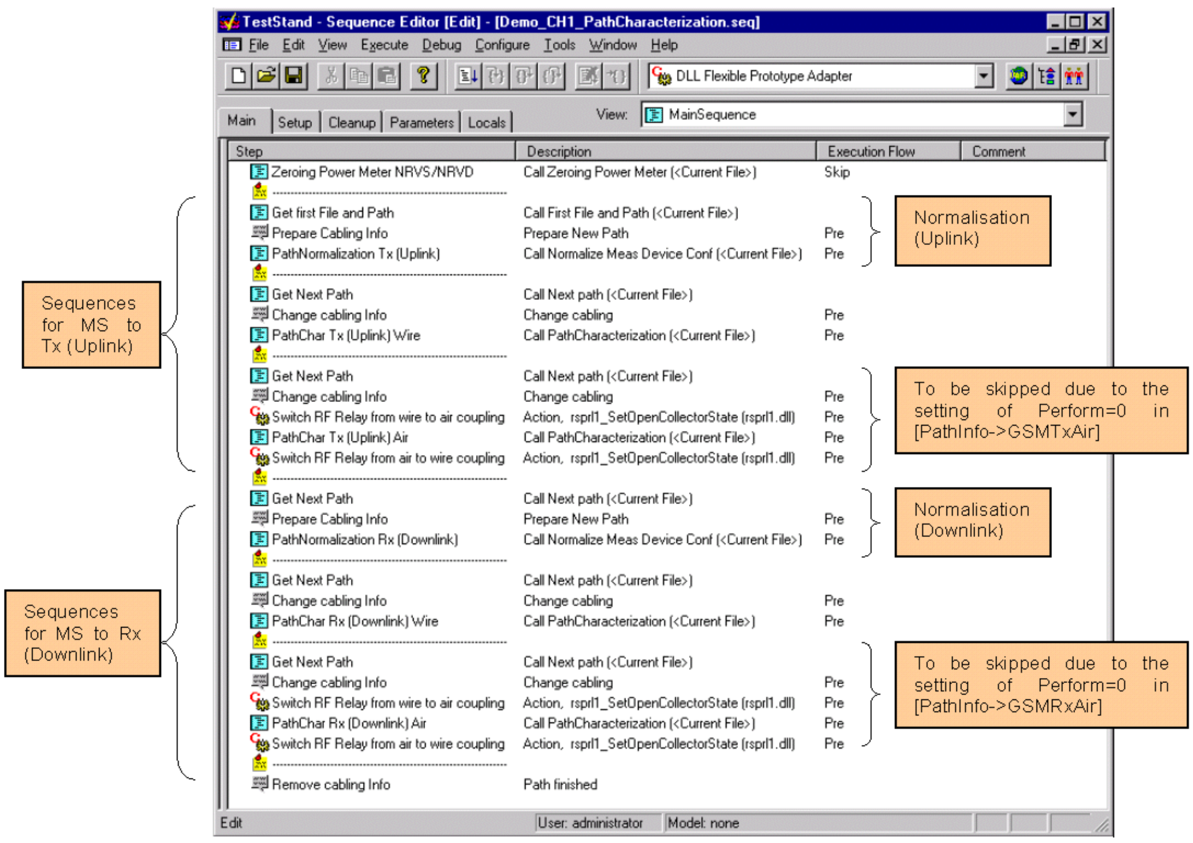
Get Next Path	Call Next path (<Current File>)		
Change cabling Info	Change cabling	Pre	
PathChar Rx (Downlink) Wire	Call PathCharacterization (<Current File>)	Pre	

Get Next Path	Call Next path (<Current File>)		
Change cabling Info	Change cabling	Pre	
Switch RF Relay from wire to air coupling	Action, rsprl1_SetOpenCollectorState (rsprl1.dll)	Pre	
PathChar Rx (Downlink) Air	Call PathCharacterization (<Current File>)	Pre	
Switch RF Relay from air to wire coupling	Action, rsprl1_SetOpenCollectorState (rsprl1.dll)	Pre	

Remove cabling Info	Path finished		

Figure 12-20 Path Characterization Testsequence

The various functions from the Path Characterization Library are integrated into the Path Characterization test sequence. The initialized Calibration File (see chapter 12.4.2) is called up with its individual sections (paths, scans) by the test sequence. This means that a relationship is created between the entries in the Calibration File and test sequence (see Figure 12-21 and Figure 12-22).


TestStand - Sequence Editor [Edit] - [Demo_CH1_PathCharacterization.seq]
File Edit View Execute Debug Configure Tools Window Help
DLL Flexible Prototype Adapter
Main Setup Cleanup Parameters Locals
View: MainSequence
Edit
User: administrator
Model: none

Step	Description	Execution Flow	Comment
Zeroing Power Meter NRVS/NRVD	Call Zeroing Power Meter (<Current File>)	Skip	
--			
Get first File and Path	Call First File and Path (<Current File>)		
Prepare Cabling Info	Prepare New Path	Pre	
PathNormalization Tx (Uplink)	Call Normalize Meas Device Conf (<Current File>)	Pre	
--			
Get Next Path	Call Next path (<Current File>)		
Change cabling Info	Change cabling	Pre	
PathChar Tx (Uplink) Wire	Call PathCharacterization (<Current File>)	Pre	
--			
Get Next Path	Call Next path (<Current File>)		
Change cabling Info	Change cabling	Pre	
Switch RF Relay from wire to air coupling	Action, rsprl1_SetOpenCollectorState (rsprl1.dll)	Pre	
PathChar Tx (Uplink) Air	Call PathCharacterization (<Current File>)	Pre	
Switch RF Relay from air to wire coupling	Action, rsprl1_SetOpenCollectorState (rsprl1.dll)	Pre	
--			
Get Next Path	Call Next path (<Current File>)		
Prepare Cabling Info	Prepare New Path	Pre	
PathNormalization Rx (Downlink)	Call Normalize Meas Device Conf (<Current File>)	Pre	
--			
Get Next Path	Call Next path (<Current File>)		
Change cabling Info	Change cabling	Pre	
PathChar Rx (Downlink) Wire	Call PathCharacterization (<Current File>)	Pre	
--			
Get Next Path	Call Next path (<Current File>)		
Change cabling Info	Change cabling	Pre	
Switch RF Relay from wire to air coupling	Action, rsprl1_SetOpenCollectorState (rsprl1.dll)	Pre	
PathChar Rx (Downlink) Air	Call PathCharacterization (<Current File>)	Pre	
Switch RF Relay from air to wire coupling	Action, rsprl1_SetOpenCollectorState (rsprl1.dll)	Pre	
--			
Remove cabling Info	Path finished		

Sequences for MS to Tx (Uplink)

Normalisation (Uplink)

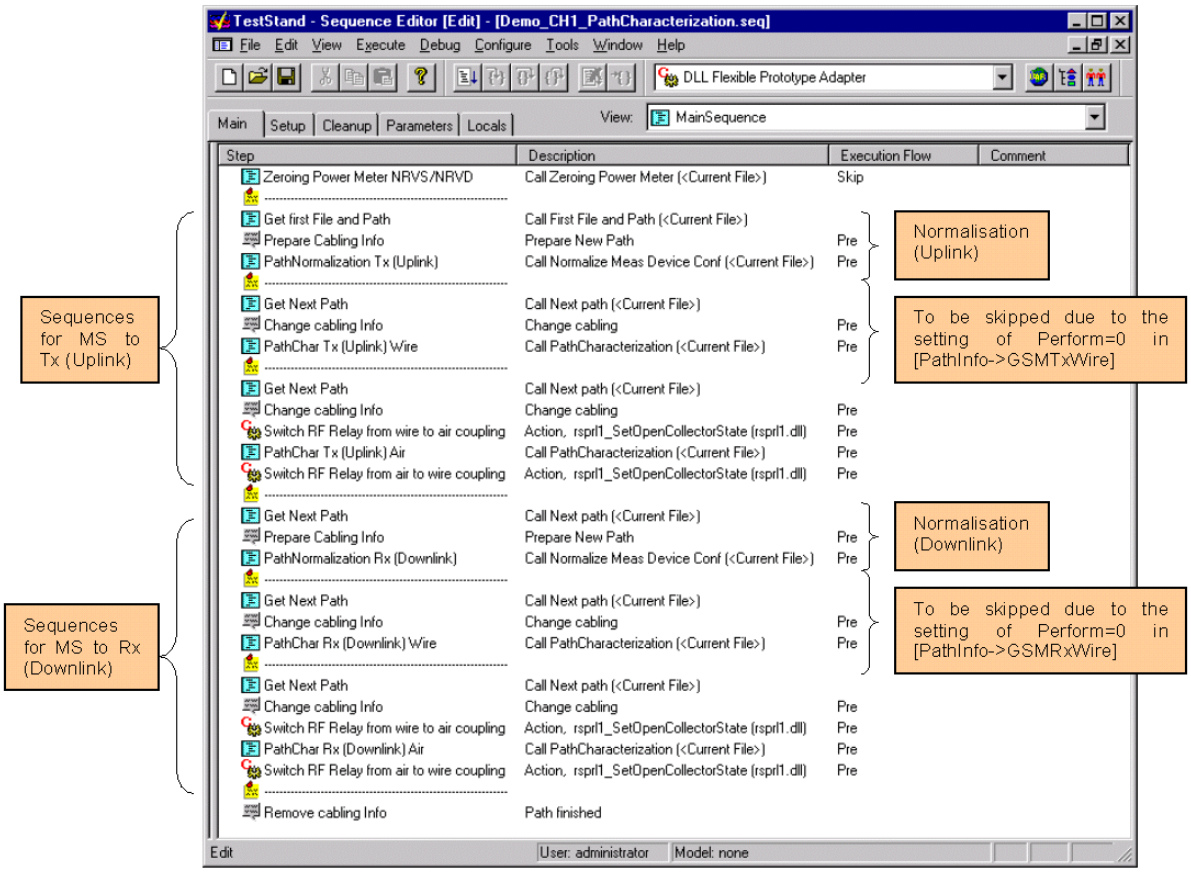
To be skipped due to the setting of Perform=0 in [PathInfo->GSMTxAir]

Sequences for MS to Rx (Downlink)

Normalisation (Downlink)

To be skipped due to the setting of Perform=0 in [PathInfo->GSMRxAir]

Figure 12-21 Path Characterization Testsequence Wire Coupling


TestStand - Sequence Editor [Edit] - [Demo_CH1_PathCharacterization.seq]
File Edit View Execute Debug Configure Tools Window Help
DLL Flexible Prototype Adapter
Main Setup Cleanup Parameters Locals
View: MainSequence
Sequences for MS to Tx (Uplink)
Sequences for MS to Rx (Downlink)
Normalisation (Uplink)
To be skipped due to the setting of Perform=0 in [PathInfo->GSMTxWire]
Normalisation (Downlink)
To be skipped due to the setting of Perform=0 in [PathInfo->GSMRxWire]
Edit
User: administrator
Model: none

Step	Description	Execution Flow	Comment
Zeroing Power Meter NRVS/NRVD	Call Zeroing Power Meter (<Current File>)	Skip	
Get first File and Path	Call First File and Path (<Current File>)		
Prepare Cabling Info	Prepare New Path	Pre	
PathNormalization Tx (Uplink)	Call Normalize Meas Device Conf (<Current File>)	Pre	
Get Next Path	Call Next path (<Current File>)		
Change cabling Info	Change cabling	Pre	
PathChar Tx (Uplink) Wire	Call PathCharacterization (<Current File>)	Pre	
Get Next Path	Call Next path (<Current File>)		
Change cabling Info	Change cabling	Pre	
Switch RF Relay from wire to air coupling	Action, rsprl1_SetOpenCollectorState (rsprl1.dll)	Pre	
PathChar Tx (Uplink) Air	Call PathCharacterization (<Current File>)	Pre	
Switch RF Relay from air to wire coupling	Action, rsprl1_SetOpenCollectorState (rsprl1.dll)	Pre	
Get Next Path	Call Next path (<Current File>)		
Prepare Cabling Info	Prepare New Path	Pre	
PathNormalization Rx (Downlink)	Call Normalize Meas Device Conf (<Current File>)	Pre	
Get Next Path	Call Next path (<Current File>)		
Change cabling Info	Change cabling	Pre	
PathChar Rx (Downlink) Wire	Call PathCharacterization (<Current File>)	Pre	
Get Next Path	Call Next path (<Current File>)		
Change cabling Info	Change cabling	Pre	
Switch RF Relay from wire to air coupling	Action, rsprl1_SetOpenCollectorState (rsprl1.dll)	Pre	
PathChar Rx (Downlink) Air	Call PathCharacterization (<Current File>)	Pre	
Switch RF Relay from air to wire coupling	Action, rsprl1_SetOpenCollectorState (rsprl1.dll)	Pre	
Remove cabling Info	Path finished		

Figure 12-22 Path Characterization Testsequence Air Coupling

Various messages are displayed during the Path Characterization test sequence. These messages contain descriptions of the individual test setups or connections, which are necessary for performing the measurements.

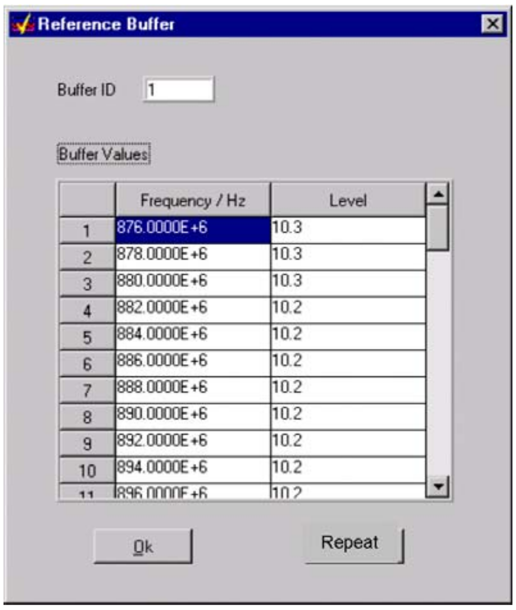
Examples:

```
Path: PathInfo NormalisationGSMTx

Connect power meter with possibly sensor cable to source instrument with possibly generator cable
directly.
```

```
Path: PathInfo->GSMTxWire

Signal Generator:
Connect SigGen with possibly source cable instead of DUT antenna connector inside fixture.

Power Meter:
Connect RF cable (from fixture)at RadioComTester side to power meter with possibly sensor cable.
```

```
.
.
.
```

Once the connection has been established and this has been confirmed by the messages, the measurements are then carried out automatically. Measurements are performed at the frequencies as given in the Calibration File. The determined measuring results are displayed in tables (see Figure 12-23 and Figure 12-24).



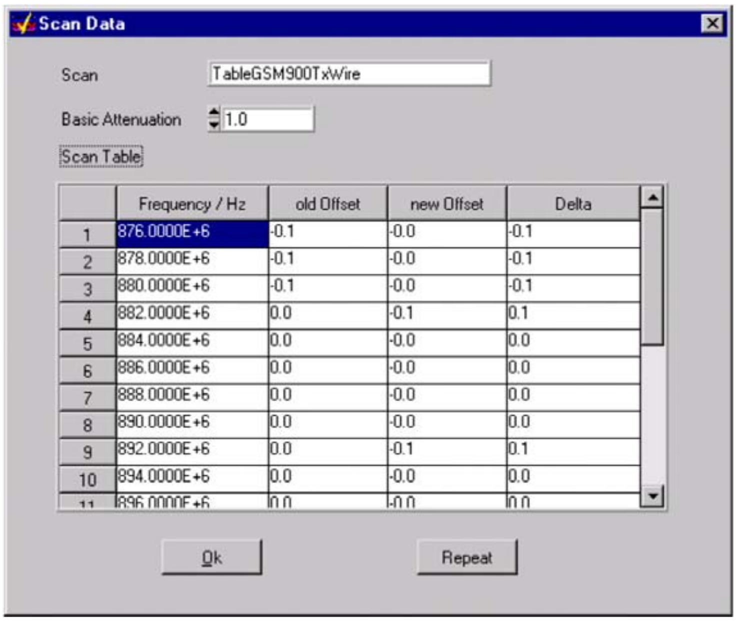
Buffer ID 1

Buffer Values

	Frequency / Hz	Level
1	876.0000E+6	10.3
2	878.0000E+6	10.3
3	880.0000E+6	10.3
4	882.0000E+6	10.2
5	884.0000E+6	10.2
6	886.0000E+6	10.2
7	888.0000E+6	10.2
8	890.0000E+6	10.2
9	892.0000E+6	10.2
10	894.0000E+6	10.2
11	896.0000E+6	10.2

Figure 12-23 Normalised Power Level

Should the test results be inaccurate or contradictory, the **Repeat** button can be used to repeat the measurement. The **OK** button serves to temporarily store the measured values for subsequent evaluation.



Scan Data

Scan: TableGSM900TxWire

Basic Attenuation: 1.0

Scan Table

	Frequency / Hz	old Offset	new Offset	Delta
1	876.0000E+6	-0.1	-0.0	-0.1
2	878.0000E+6	-0.1	-0.0	-0.1
3	880.0000E+6	-0.1	-0.0	-0.1
4	882.0000E+6	0.0	-0.1	0.1
5	884.0000E+6	0.0	-0.0	0.0
6	886.0000E+6	0.0	-0.0	0.0
7	888.0000E+6	0.0	-0.0	0.0
8	890.0000E+6	0.0	-0.0	0.0
9	892.0000E+6	0.0	-0.1	0.1
10	894.0000E+6	0.0	-0.0	0.0
11	896.0000E+6	0.0	-0.0	0.0

Figure 12-24 Measured Correction Values

Here too, should the test results be inaccurate or contradictory, the **Repeat** button can be used to repeat the measurement. Using the **OK** button will save the measured values. A basic attenuation can be specified at this point for the measured values.

The stored values and the basic attenuation are written into the invoked Calibration File at the end of the Path Characterization test sequence.

12.4.4 Typical Test Setups (Examples)

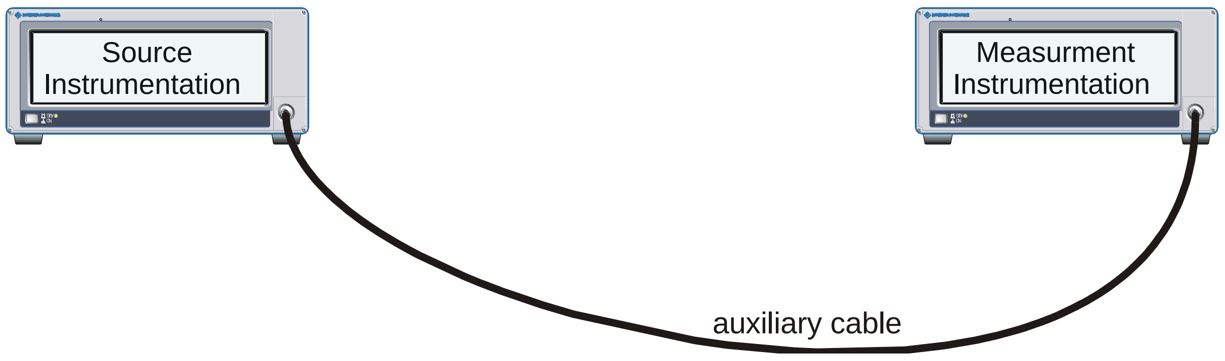


Figure 12-25 Test Setup Normalization for Uplink (Tx) and Downlink (Rx) (Example)

The auxiliary cable is measured during the normalization test and used for the subsequent test setups.

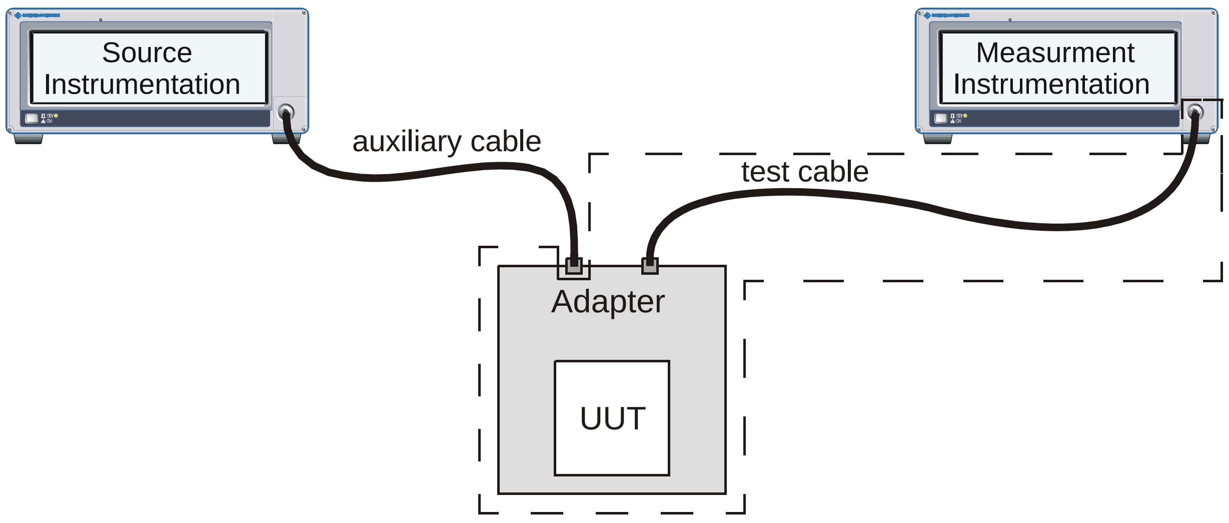


Figure 12-26 Path Characterization Test Setup for Uplink (Tx) (Example)

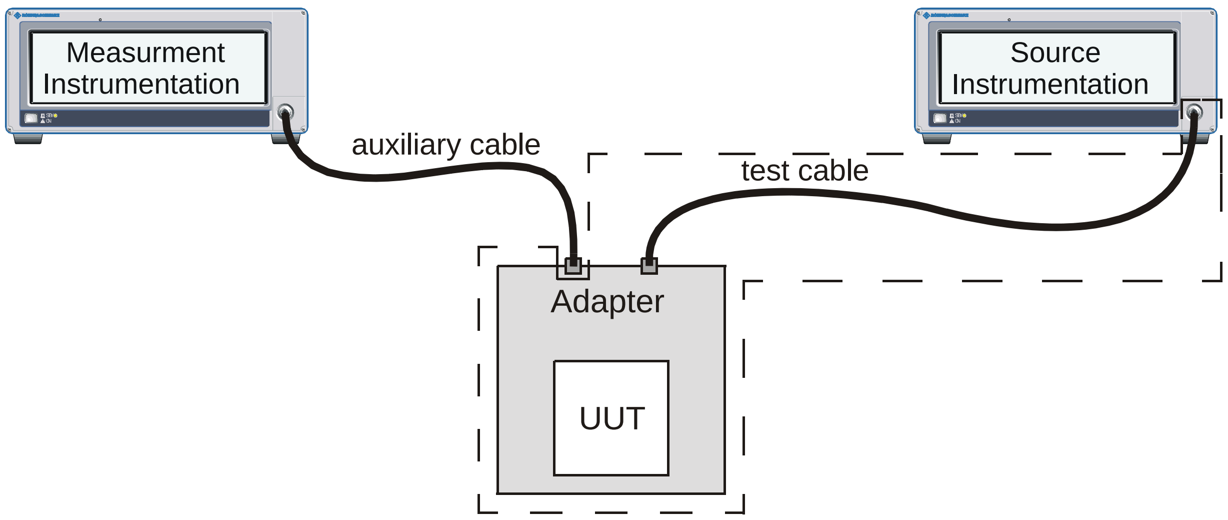


Figure 12-27 Path Characterization Test Setup for Downlink (Tx) (Example)

The auxiliary cable is used in the two measurements in Figure 12-26 and Figure 12-27. Now the correction values for the signal path are measured (dotted area: test cable + adapter). The temporarily stored measured values for the auxiliary cable are automatically removed from the calculated values for the Calibration File.

13 Instrument Soft Panels

The Instrument Soft Panels permit interactive operation of all TSVP hardware modules. The Soft Panels can be used to perform all the setting, switching and measuring functions.

In addition, they offer a range of useful tools, such as:

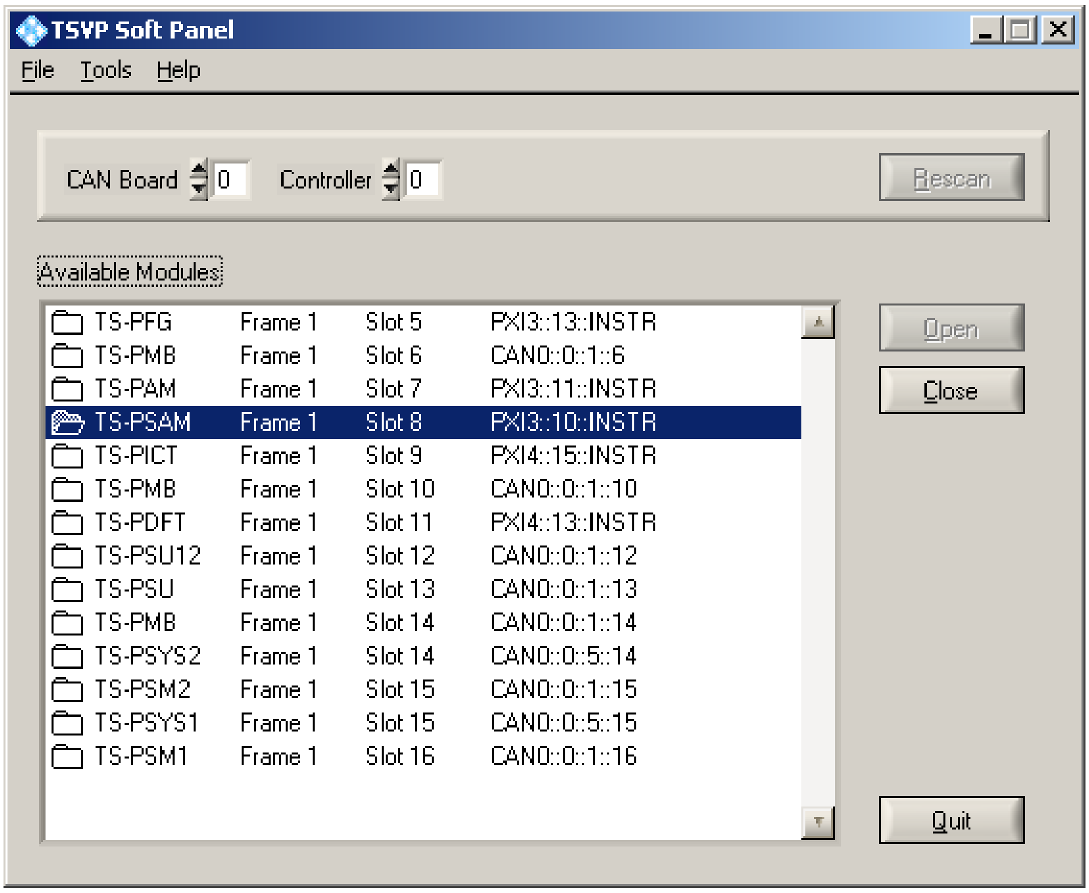
- **Pin Location**: Using this tool, you can verify the correct wiring and contacting of a test adapter.
- **Create Physical.ini**: Tool for automatically creating a "`Physical.ini`" file.

13.1 Starting the Soft Panels

Proceed as follows to start the Soft Panels:

1. Start the TSVP Soft Panel application via the menu path **Start -> Programs -> GTSL -> Instrument Soft Panels**

   First, the software will determine the modules available in the system and display them:



TS-PFG	Frame 1	Slot 5	PXI3::13::INSTR
TS-PMB	Frame 1	Slot 6	CAN0::0::1::6
TS-PAM	Frame 1	Slot 7	PXI3::11::INSTR
TS-PSAM	Frame 1	Slot 8	PXI3::10::INSTR
TS-PICT	Frame 1	Slot 9	PXI4::15::INSTR
TS-PMB	Frame 1	Slot 10	CAN0::0::1::10
TS-PDFT	Frame 1	Slot 11	PXI4::13::INSTR
TS-PSU12	Frame 1	Slot 12	CAN0::0::1::12
TS-PSU	Frame 1	Slot 13	CAN0::0::1::13
TS-PMB	Frame 1	Slot 14	CAN0::0::1::14
TS-PSYS2	Frame 1	Slot 14	CAN0::0::5::14
TS-PSM2	Frame 1	Slot 15	CAN0::0::1::15
TS-PSYS1	Frame 1	Slot 15	CAN0::0::5::15
TS-PSM1	Frame 1	Slot 16	CAN0::0::1::16

Figure 13-1 TSVP Soft Panel, main window

2. Select a module in the list and click **Open**. The instrument panel for the module is displayed.

   The module can only be operated via the instrument panel. In the main window you can start other instrument panels at any time.

☞

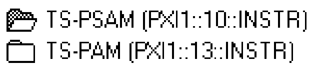
NOTE:

Once the Instrument Soft Panels have been started, no other application must be active that also require the hardware modules, such as the self-test or a test application.
If hardware modules are not found, theTSVP Soft Panel displays a simulation module for each TSVP module type.

13.2 Main Window

13.2.1 Controls

The main window displays all the TSVP modules available in the system. The associated instrument panels are started via the **Open** button or simply by doubleclicking the desired list entry.


TS-PSAM (PXI1::10::INSTR)
TS-PAM (PXI1::13::INSTR)

An already open instrument panel is represented by an open 'folder'. Doubleclick a list entry of that type to move the panel to the foreground.

Click the **Close** button to close an open panel.

The **Quit** button terminates the TSVP Soft Panel and closes any open instrument panels.

The **CAN Board** and **Controller** settings refer to the configuration of the CAN bus for the TSVP modules. The default value for these settings is 0. For special system configurations, a different CAN controller can be selected in these fields. The **Rescan** button triggers a new search for CAN modules in the system.

13.2.2 Menus

The **<File><Exit>** menu command terminates the TSVP Soft Panel and closes any open instrument panels.

The **<Tools>** menu provides various help programs such as automatically creating a “`Physical.ini`” file that describes the hardware configuration of the R&S CompactTSVP. For more information on this topic, please refer to chapter 13.4 in this manual.

The **<Help><Usage...>** menu command provides information on the command line parameters of the Soft Panel, please refer to the following chapter.

The **<Help><About>** menu command displays the version number of the TSVP Soft Panel and of the R&S GTSL software currently used on the system.

13.2.3 Command Line Parameters

The TSVP Soft Panel can be started with the following command line parameters:

-simulation
In addition to the hardware modules found, a simulation module is displayed for each TSVP module type. It permits operation in simulation mode, i.e. without any physical hardware present.

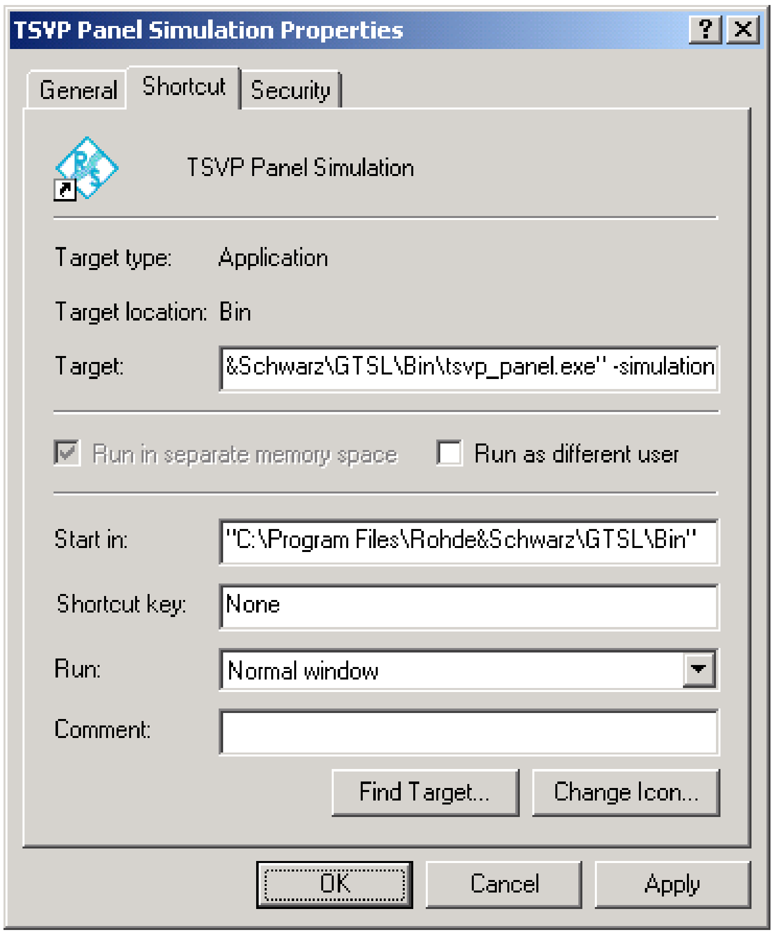
-nocan
The CAN bus is not scanned upon start of the TSVP Soft Panel. This option accelerates the start of the Soft Panel if there are no CAN modules.

-nopxi
The PCI/PXI bus is not scanned upon start of the TSVP Soft Panel. This option accelerates the start of the Soft Panel if there are no PCI modules.

-can<b>::<c>
This parameter is used for pre-assigning the settings **CAN Board** and **Controller**, where <b> stands for theCAN board number and <c> for the controller number. This option is useful if the CAN bus is not controlled via the standard controller. Example: -can1::1

The TSVP Soft Panel can be started with command line parameters in two ways:

1. By opening a prompt with **Start -> Programs -> Accessories -> Command Prompt** and entering a call, e.g.
   `C:\> tsvp_panel -simulation`
2. By creating a shortcut on the desktop:
   a) Right-click the desktop and select the menu command **New -> Shortcut**. In the following dialog, select the file `C:\Program Files\Rohde&Schwarz\GTSL\Bin\tsvp_panel.exe`. In the next step, assign a name to the shortcut, e.g. “TSVP Panel Simulation”.
   b) Right-click the new shortcut and select the menu command **Properties**.


TSVP Panel Simulation Properties

TSVP Panel Simulation

Target type: Application

Target location: Bin

Target: &Schwarz\GTSL\Bin\tsvp_panel.exe" -simulation

[✓] Run in separate memory space
[ ] Run as different user

Start in: "C:\Program Files\Rohde&Schwarz\GTSL\Bin"

Shortcut key: None

Run: Normal window

Comment:

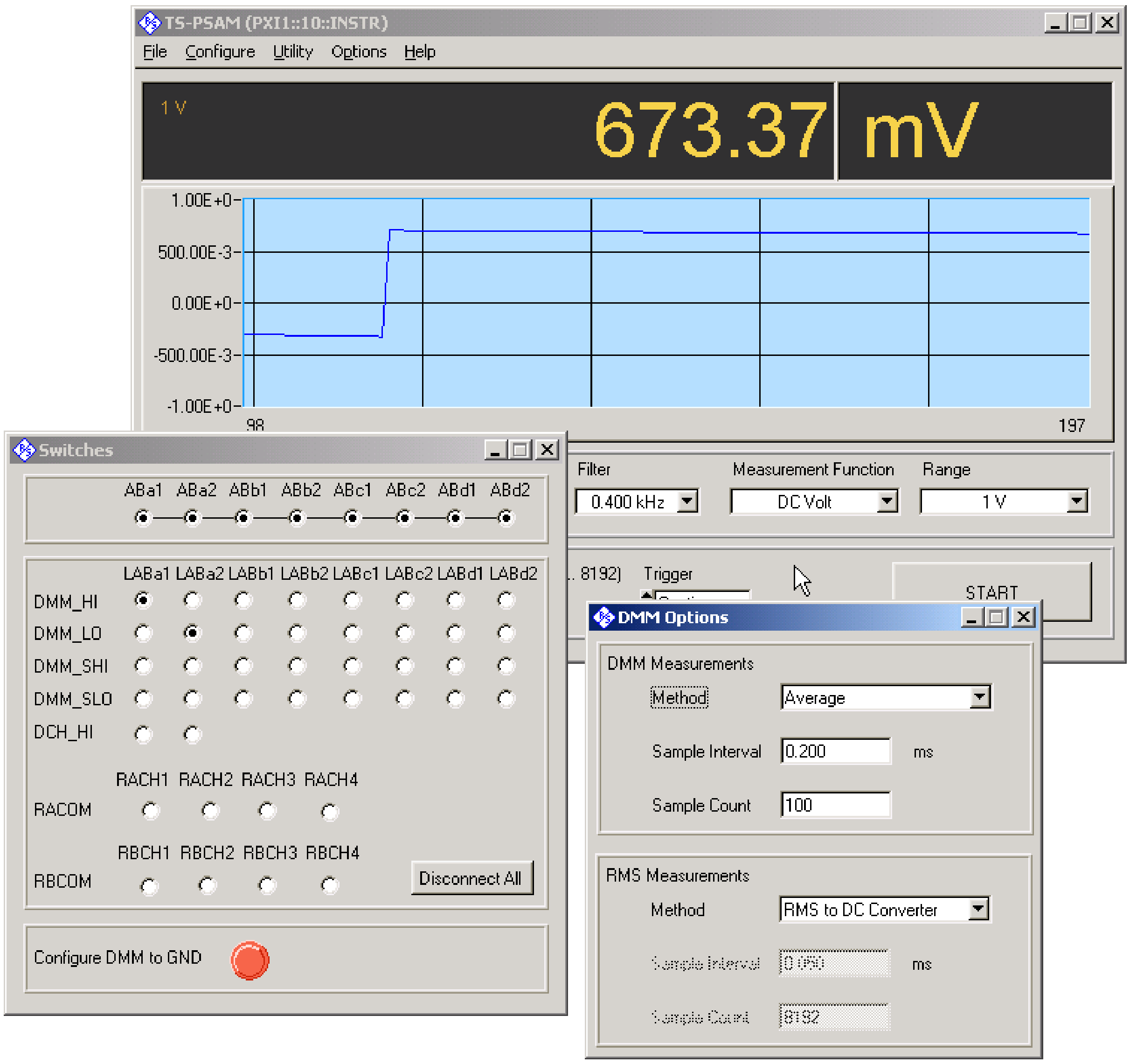
Figure 13-2 Shortcut to TSVP Soft Panel

c) Now enter the command line parameter(s) in the “Target” field behind the file name.

d) Doubleclick the new shortcut to start the TSVP Soft Panel with these new command line parameters.

13.3 Instrument Panels

The individual instrument panels permit interactive operation of the respective module, such as setting, switching and measuring. An instrument panel is made up of a **main window** accommodating the most frequently used controls. Further subdialog windows and functions can be called via **menus**.



Filter 0.400 kHz
Measurement Function DC Volt
Range 1 V

DMM Measurements
Method Average
Sample Interval 0.200 ms
Sample Count 100

RMS Measurements
Method RMS to DC Converter
Sample Interval 0.050 ms
Sample Count 8192

Figure 13-3 R&S TS-PSAM instrument panel with subdialog windows

The instrument panels feature a similar design. For this reason, the following chapters describe the properties common to all the instrument panels.

13.3.1 Menu Structure

The **<File><Close>** menu command closes the instrument panel.

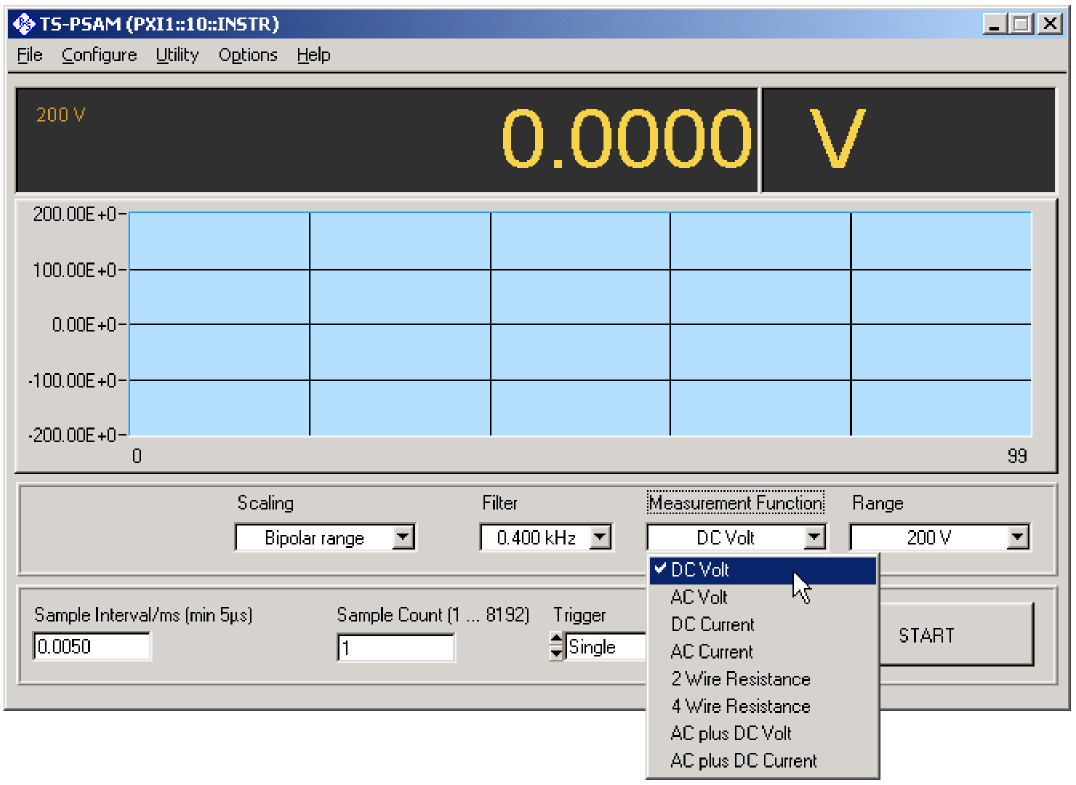
The **<Configure>** menu command provides a number of additional menu commands for displaying the subdialog windows for the switching, triggering and the like.

The **<Utility><Revision Query...>** menu command displays information on the serial number, the firmware version and the software version of the module.

The **<Utility><Reset>** menu command resets the module to the default setting.

The **<Help><About>** menu command displays the version number of the TSVP Soft Panel and of the R&S GTSL software currently used on the system.

13.3.2 Settings


TS-PSAM (PXI1::10::INSTR)

200 V 0.0000 V

Scaling: Bipolar range
Filter: 0.400 kHz
Measurement Function: DC Volt
Range: 200 V

Sample Interval/ms (min 5µs): 0.0050
Sample Count (1 ... 8192): 1
Trigger: Single

Figure 13-4 Settings, R&S TS-PSAM

There are various input possibilities for configuring the modules.

- Drop-down list boxes for selecting individual options, e.g. the Measurement Funcion as shown in Figure 13-4.
- Text input fields for numeric values.
- Buttons and slide controls.

The setting is effective as soon as the entry has been made. Subdialog boxes with an **Apply** button are an exception. In this case, the data are accepted only when **Apply** is pressed.

13.3.3 Subdialog Window

A subdialog window offers extended settings such as Triggering that can be called via the main window menu (Figure 13-4). Figure 13-5 shows a submenu window for setting the triggering.

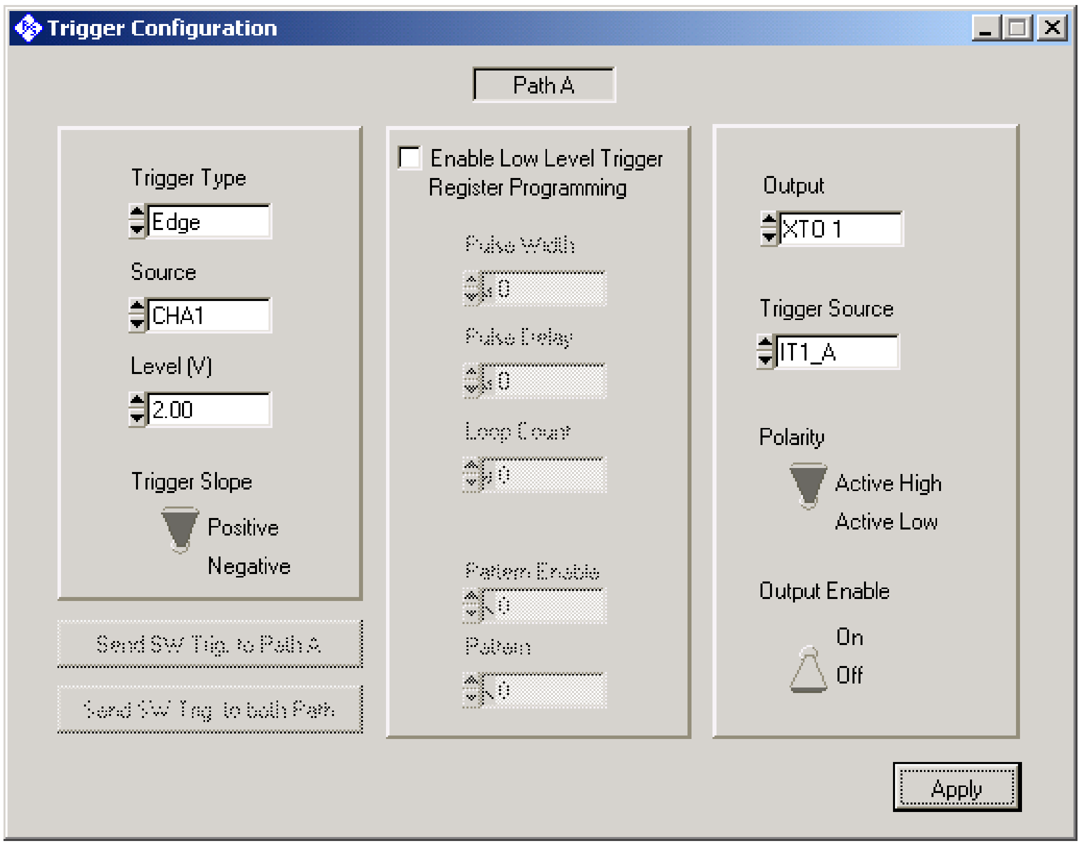


Figure 13-5 Subdialog window, R&S TS-PAM trigger setting

The subdialogs are called via the **<Configure>** menu. The subdialogs are displayed in parallel to the main window of the instrument panel. If a subdialog window has an **Apply** button, the data will be sent to the instrument only when **Apply** has been clicked.

Subdialog windows are closed by clicking the ☒ button in the title bar.

13.3.4 Relay Matrix

Depending on the module, the connections are either made in the main window or via a subdialog window in the **Configure><Switches...>** menu.

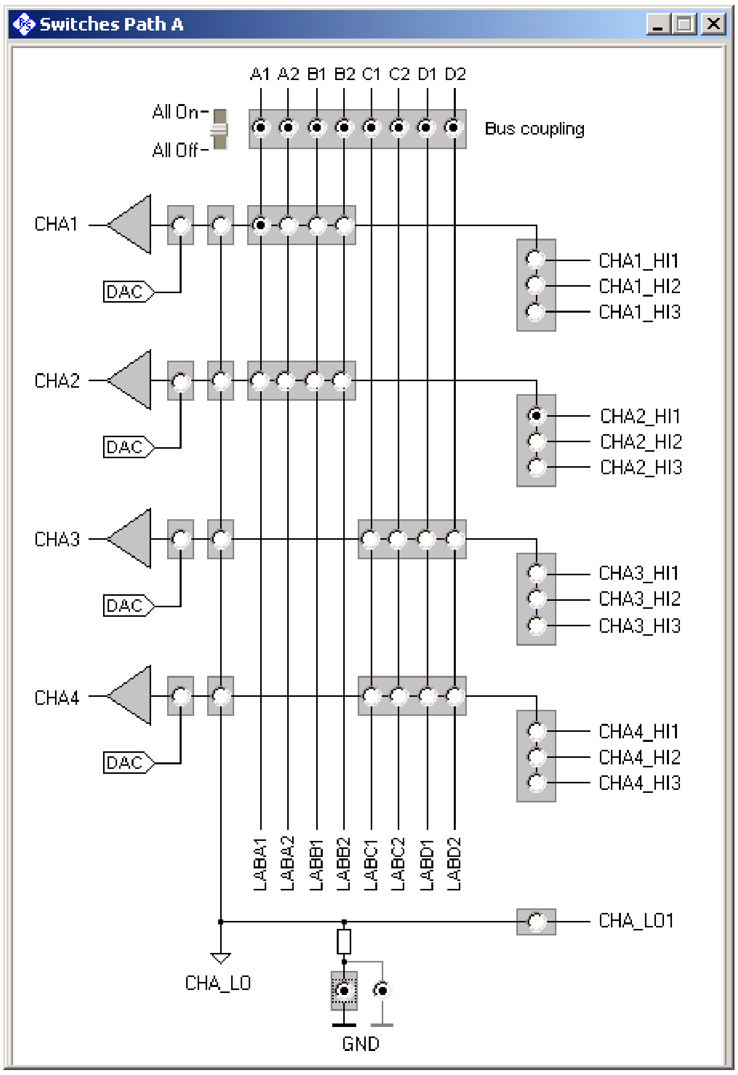


Figure 13-6 Relay matrix, R&S TS-PAM

The relay matrix of the modules to the analog bus and the front connector is displayed in the form of a graphics. The controls at the junctions of the lines are relays. The relays can be opened and closed by clicking the controls. Closed relays are represented with a dot.

13.4 Tools

The following software tools can be accessed via the **<Tools>** menu of the TSVP Soft Panel:

<Tools><Pin Location...>

Tool for checking the adapter wiring. Refer to chapter 13.4.1.

<Tools><Create Physical.ini...>

Tool for automatically creating a "`Physical.ini`" file for the current system. Refer to chapter 13.4.2.

<Tools><Front Connectors>

Tool for displaying module front connectors. Refer to chapter 13.4.3.

NOTE:

The tools can be called only if an instrument panel is not open.

13.4.1 Pin Location

Pin Location is used to verify the adapter wiring using a probe. A pin can be identified rapidly by touching it with a probe. This tool is also useful in the case of contact problems.

13.4.1.1 Hardware required

Pin Location requires a Source and Measurement Module R&S TS-PSAM for measuring the resistance and one or more R&S TS-PMB matrix modules to which the unit under test (UUT) or adapter is connected.

13.4.1.2 Connecting the Probe

The probe is directly connected to the front connector of the R&S TS-PSAM module. The following connections are possible:

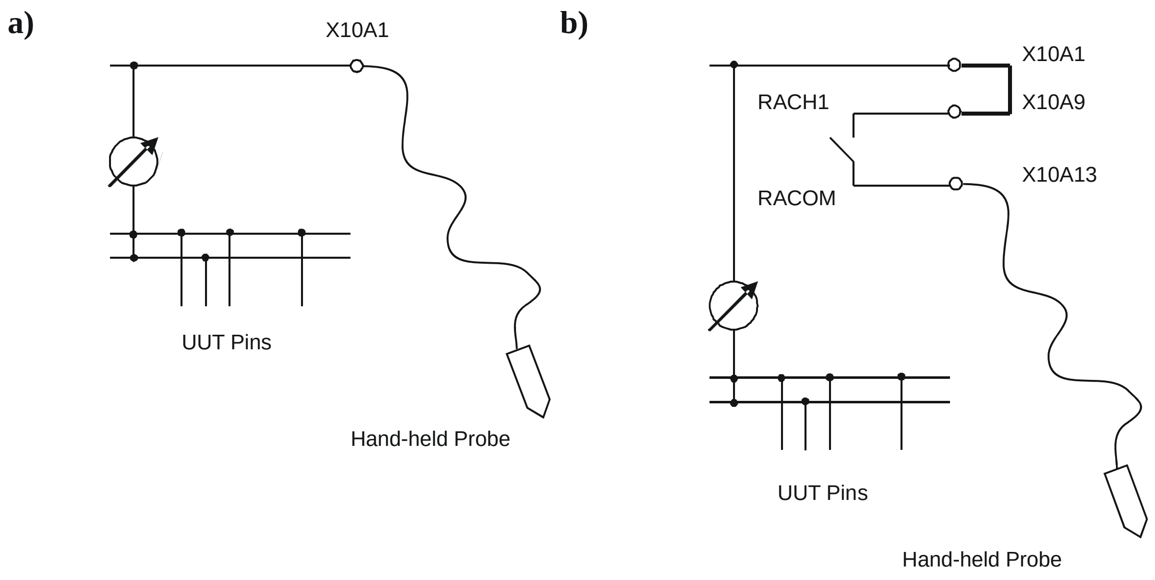


Figure 13-7 Connecting the probe (example)

Option **a)** is suited primarily for a “fast” verification of the adapter wiring. It merely requires a cable with a probe.

NOTE:

The cable must be disconnected again as soon as the test program or another application (e.g. the self-test) is started. Otherwise faulty measurements may result since the probe is permanently connected to the ABa1 analog bus.

Option **b)** should always be selected when the probe or a socket for the probe is integrated in the adapter. In this case the probe is disconnected from analog bus ABa1 via a relay multiplexer of the R&S TS-PSAM module when it is not being used. For this purpose, the adapter must be fitted with an additional wire bridge.

13.4.1.3 Measurement Principle

After starting Pin Location, the software first discharges the pins to be tested individually towards each other and towards ground. If the residual voltage of a pin is still too high after it has been discharged, it cannot be included in the scan list. If the highest residual voltage measured exceeds 5 V, Pin Location cannot be started since the measuring system could be at risk with this setting.

After discharging, the software configures the R&S TS-PSAM module as an Ohmmeter and the DMM_HI connector is coupled with the probe via the local analog bus LABa1.

The DMM_LO connector is coupled to all the UUT pins, i.e. they are short-circuited towards each other. The Ohmmeter continuously measures the resistance. As long as the probe does not have contact to one of the test pins, the measurement will yield a high-resistance result.

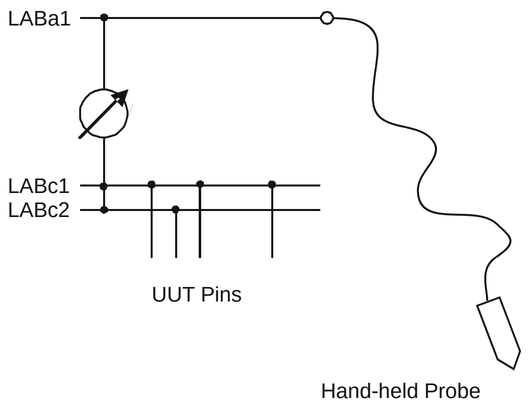


Figure 13-8 Measurement Principle

As soon as the probe touches a test pin, the measurement will supply a low resistance that depends on the resistances of the matrix relays, the supply lines and the contact resistance of the probe. As soon as this resistance is lower than 10 Ohm, the actual scan is started. This requires that the contact remains until the pin has been located.

For the scan, first all the test pins are disconnected from the DMM_LO connector of the Ohmmeter. Thereafter, they are individually connected to DMM_LO one after the other, and the resistance is measured again. If a low value is measured here, the connected pin has been found and the information is output.

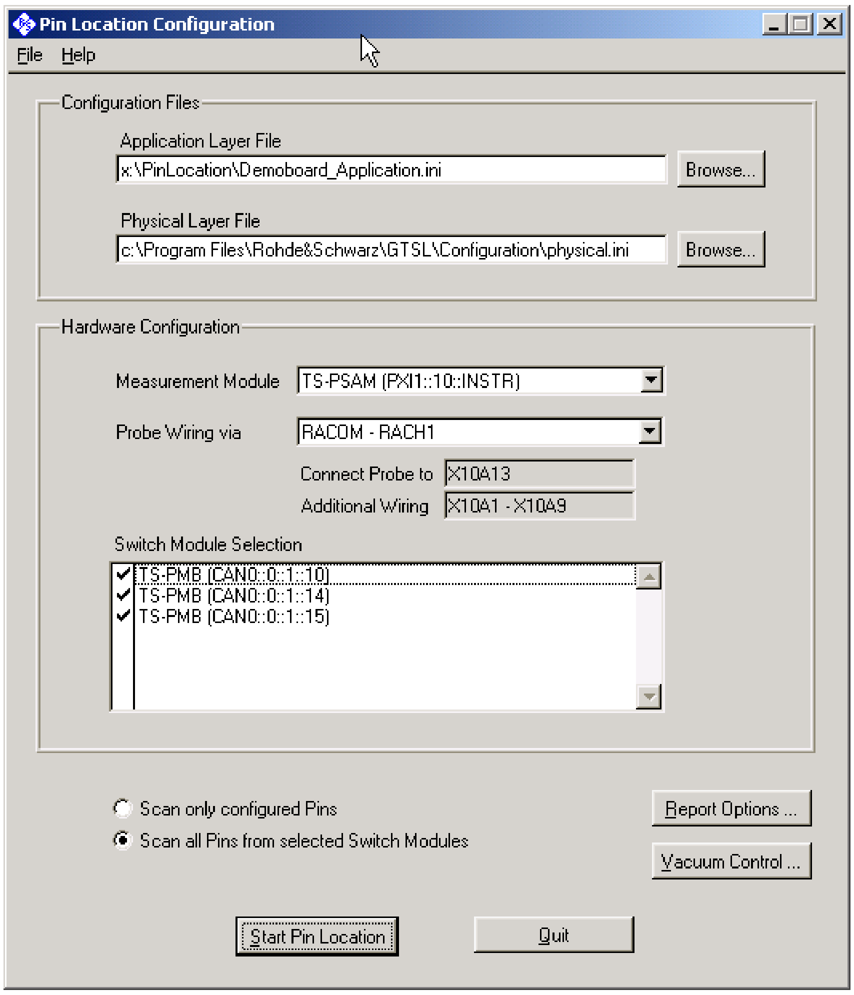
As soon as the scan has been completed, all the pins are reconnected to DMM_LO and the program waits for another contact.

13.4.1.4 Starting Pin Location

Pin Location is started via the menu command **<Tools><Pin Location...>** of the TSVP Soft Panel or via function key **F2**.

13.4.1.5 Configuration Dialog

The Configuration dialog is displayed after Pin Location has been started (Figure 13-9).



Application Layer File: x:\PinLocation\Demoboard_Application.ini

Physical Layer File: c:\Program Files\Rohde&Schwarz\GTSL\Configuration\physical.ini

Measurement Module: TS-PSAM (PXI1::10::INSTR)

Probe Wiring via: RACOM - RACH1

Connect Probe to: X10A13

Additional Wiring: X10A1 - X10A9

- ✓ TS-PMB (CAN0::0::1::10)
- ✓ TS-PMB (CAN0::0::1::14)
- ✓ TS-PMB (CAN0::0::1::15)

○ Scan only configured Pins
● Scan all Pins from selected Switch Modules

Figure 13-9 Configuration Dialog

The following settings can be made in this dialog:

Configuration Files window area:

In this window area, the configuration files for the physical and application layers can optionally be entered. These files are required to be able to display the logical node names (from `application.ini`) and the device names (from `physical.ini`).
Use the **Browse...** button to select the files.

Application Layer File In this field, enter the path and the name of the `application.ini` file, in which the name assignment of the adapter to be tested is stored. Pin Location reads all the name tables (i.e. sections starting with [io_channel->]) from this file and uses them for displaying the logical node names. If a file is not specified here, the logical node names are not converted and only the physical names will be displayed.

Physical Layer File In this field, enter the path and the name of the `physical.ini` file, in which the configuration of the test system is stored. By default, this is the file “`physical.ini`” in the folder “`C:\Program Files\Rohde&Schwarz\GTSL\Configuration`”. If a file is not specified in this field, the device names are not converted. The file must always be specified, when an `application.ini` file is specified.

Hardware Configuration window area

The hardware modules and the probe wiring are selected in this window area.

Measurement Module Select the R&S TS-PSAM module that is to be used for the measurements and to which the probe is to be connected.

Probe Wiring via Select how the probe is to be connected. The fields below this field show, to which pins of the front connector the probe and the bridge must be connected.
The following connection possibilities have been provided:

Connection via	Connection of the probe	Connection of the bridge
LABA1	X10 A 1	NA
RACOM - RACH1	X10 A 13	X10 A 1 - X10 A 9
RACOM - RACH2	X10 A 13	X10 A 1 - X10 A 10
RACOM - RACH3	X10 A 13	X10 A 1 - X10 A 11
RACOM - RACH4	X10 A 13	X10 A 1 - X10 A 12
RBCOM - RBCH1	X10 B 13	X10 A 1 - X10 B 9
RBCOM - RBCH2	X10 B 13	X10 A 1 - X10 B 10
RBCOM - RBCH3	X10 B 13	X10 A 1 - X10 B 11
RBCOM - RBCH4	X10 B 13	X10 A 1 - X10 B 12

Switch Module Selection

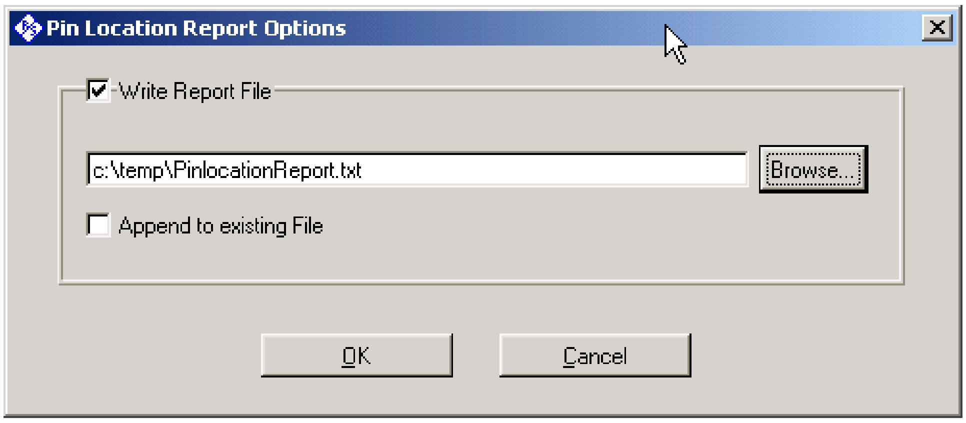
Select the switch module(s) to be included in the scan. By default, all the switch modules are selected.

Scan Options

The option **Scan only configured Pins** limits the scan to those pins that are listed in a name table in the Application Layer File. You can narrow down the scan further by delecting swith modules in the Switch Module Selection field. This option is available only if an Application Layer File has been specified.
The option **Scan all Pins from selected Switch Modules** lets Pin Location perform a scan across all the pins of the selected switch modules.

13.4.1.6 Report Options

This dialog permits the recording of a report file. The report format is described in chapter 13.4.1.10.



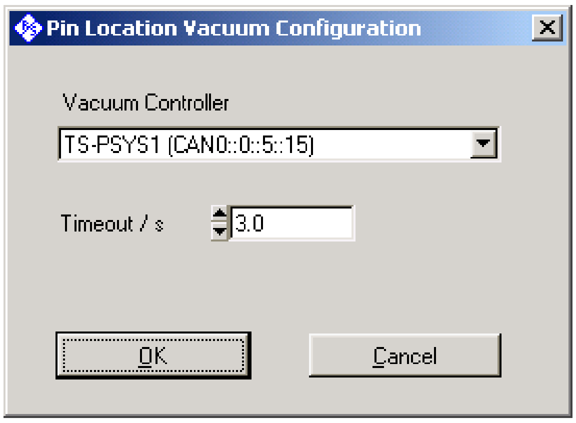
c:\temp\PinlocationReport.txt

Figure 13-10 Report Options

Write Report File	Enable this checkbox if you want to create a report file. Enter the path and file name for the report file. Use the **Browse...** button to select the path.
Append to existing File	Enable this checkbox if you want to append the report to an existing report. Otherwise, any existing report will be overwritten after a warning prompt. Confirm the dialog by clicking the **OK** button to create the report file.

13.4.1.7 Vacuum Controller

This dialog permits the selection of a R&S TS-PVAC vacum controller.


Pin Location Vacuum Configuration

Vacuum Controller: TS-PSYS1 (CAN0::0::5::15)

Timeout / s: 3.0

Figure 13-11 Vacuum Controller

Vacuum Controller Select a R&S TS-PSYS module in the list for controlling the vacuum controller or select (**no vacuum**) if there is no vacuum controller.

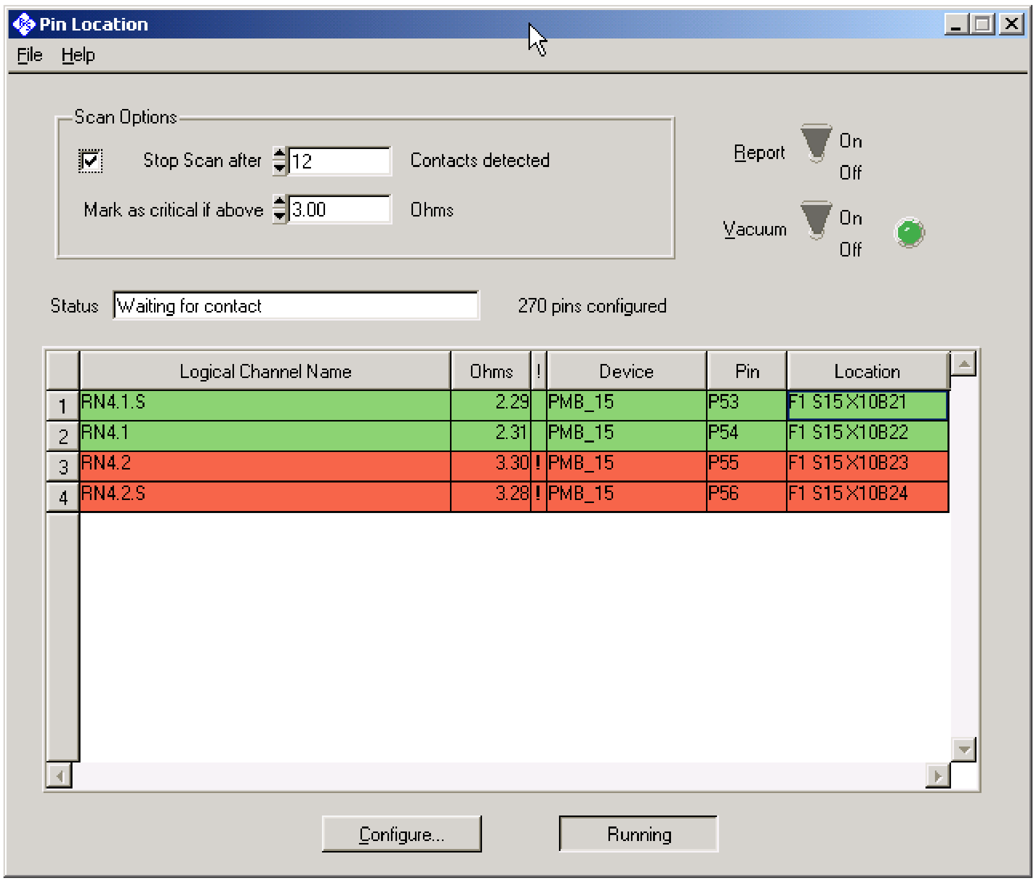
Timeout / s Enter the maximum wait time in seconds for the maximum permissible interval between the actuation of the vacuum valve and the 'Switch closed' feedback, before an error is reported.
When you confirm the dialog by clicking the **OK** button, the vacuum is not yet enabled. This is done only directly before the scanning procedure is started.

13.4.1.8 Starting the Scanning Procedure

The **Start Pin Location** button starts the scanning procedure. First, the modules are initiated and the vacuum is enabled. Thereafter, the discharge procedure is started. Once all the pins have a potential of zero, the measurement dialog will be displayed.

13.4.1.9 Measurement Dialog

The measurement dialog shows the status of the scanning procedure and the pins located, and it offers various options.



[✓] Stop Scan after 12 Contacts detected

Mark as critical if above 3.00 Ohms

Status: Waiting for contact

270 pins configured

	Logical Channel Name	Ohms	!	Device	Pin	Location
1	RN4.1.S	2.29		PMB_15	P53	F1 S15 X10B21
2	RN4.1	2.31		PMB_15	P54	F1 S15 X10B22
3	RN4.2	3.30	!	PMB_15	P55	F1 S15 X10B23
4	RN4.2.S	3.28	!	PMB_15	P56	F1 S15 X10B24

Figure 13-12 Measurement dialog

Table of Contacts

The display of the contacts located takes up the main part of the dialog. For each contact, it shows the logical channel name, the value measured, the name of the switch module, the physical pin name and the designation of the location on the front connector of the TSVP system.
Example: F1 S15 X10B21 means:

• F1	TSVP frame 1
• S15	Slot 15
• X10B15	Connector X10, column B, row 15

Some of the fields of the table will remain empty if a configuration file is not specified.

- Logical Name is empty if no `Application.ini` file is specified or if the pin is not referenced in any name table.
- Device is empty if no `Physical.ini` file is specified or if the switch module is not contained in the `Physical.ini` file.

Status

The status is shown above the table (Discharging, Waiting for Contact, Scanning, n Pins found, Paused), next to it the number of pins configured for the scan operation.

Scan Options

The option **Stop scan after n Contacts detected** aborts a scan as soon as the number of contacts specified has been detected. This can reduce the scanning period. When the checkbox is disabled, all the configured pins will be scanned.
The option **Mark as critical if above n Ohms** marks a contact as “critical”, if its value exceeds the threshold specified. It will have a red background in the table and will be identified with an exclamation mark in the table as well as in the report.

Report

This switch is used to enable/disable the recording of the report. The switch is disabled if a report was not created.

Vacuum

This switch is used to enable/disable the vacuum. The LED to the right shows the current vacuum status.

The **Running** button is used to pause the contact measurement, e.g. to correct the wiring in the adapter. In this case, the caption will change to **Paused**. Click the button again to resume the scan operation.

The **Configure** buton closes the measurement dialog and the configuration dialog is displayed again.

Menus

Use the menu command **<File><Exit>** to exit Pin Location.
The **<Help><About>** menu command displays the version number of the TSVP Soft Panel and of the R&S GTSL software currently used on the system.

13.4.1.10 Report Format

The following is an example of a Pin Location report.

```
Pin Location started at 2005-07-05 16:51:35

Configuration Files
  Application : x:\PinLocation\Demoboard_Application.ini
  Physical    : c:\Program Files\Rohde&Schwarz\GTSL\Configuration\physical.ini

Measurement Module
  TS-PSAM (PXI1::10::INSTR)
  Probe connected via RACOM - RACH1

Switch Modules
  TS-PMB (CAN0::0::1::10)
  TS-PMB (CAN0::0::1::14)
  TS-PMB (CAN0::0::1::15)

Options
  Scan all pins from selected switch modules

Discharge 270 pins

Scan 270 pins
  Critical resistance : 2 Ohms

                         Logical Name  Resistance   Physical Name  Location
================================================================================
-                               RN5.3  0.98 Ohms    PMB_15!P48  F1 S15 X10B16
-                               RN5.4  0.93 Ohms    PMB_15!P49  F1 S15 X10B17
-                               RN5.5  0.97 Ohms    PMB_15!P50  F1 S15 X10B18
-                              CTRQ0R  0.97 Ohms    PMB_15!P72  F1 S15 X10C8
-                              CTRQ2R  0.95 Ohms    PMB_15!P74  F1 S15 X10C10
-                               CTRTC  0.97 Ohms    PMB_15!P77  F1 S15 X10C13
-                               RN5.3  0.95 Ohms    PMB_15!P48  F1 S15 X10B16
-                               RN5.5  0.96 Ohms    PMB_15!P50  F1 S15 X10B18
-                                  nc  1.00 Ohms    PMB_15!P52  F1 S15 X10B20
-                             RN4.1.S  1.15 Ohms    PMB_15!P53  F1 S15 X10B21
+                               RN4.1  0.98 Ohms    PMB_15!P54  F1 S15 X10B22
+                               RN4.2  2.12 Ohms !  PMB_15!P55  F1 S15 X10B23
+                             RN4.2.S  2.12 Ohms !  PMB_15!P56  F1 S15 X10B24

Pin Location finished at 2005-07-05 16:58:18
```

At the beginning of the report, the configuration is described and any problems with the discharging of the pins. Thereafter a list of the pins detected is displayed in the form of a table.

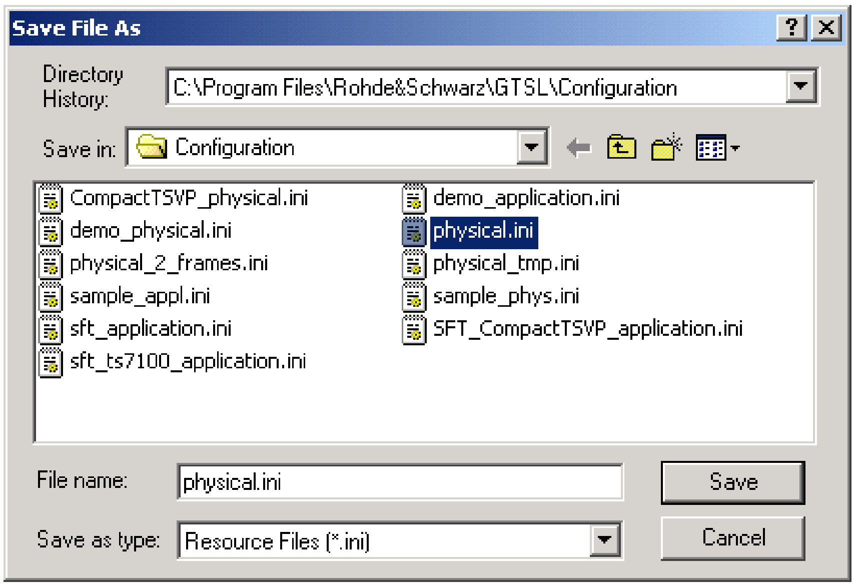
Lines beginning with “-”, identify the beginning of a new scan. Subsequent lines with “+” contain further pins detected in the same scan operation.

Resistance values exceeding the critical resistance are identified with a “!” at the end.

13.4.2 Create Physical.ini

Using the **<Tools><Create Physical.ini...>** menu item, you can create a configuration file `Physical.ini` for the current system configuration. This is useful if new modules are added or if the slot assignments were changed.

A dialog field is displayed prompting you to select the destination path.



C:\Program Files\Rohde&Schwarz\GTSL\Configuration

Configuration

CompactTSVP_physical.ini
demo_physical.ini
physical_2_frames.ini
sample_appl.ini
sft_application.ini
sft_ts7100_application.ini
demo_application.ini
physical.ini
physical_tmp.ini
sample_phys.ini
SFT_CompactTSVP_application.ini

File name: physical.ini

Save as type: Resource Files (*.ini)

Figure 13-13 Saving `Physical.ini`

Select the desired destination path for saving the new file to be created. By default, the path `C:\Program Files\Rohde&Schwarz\GT-SL\Configuration` is set.

NOTE:

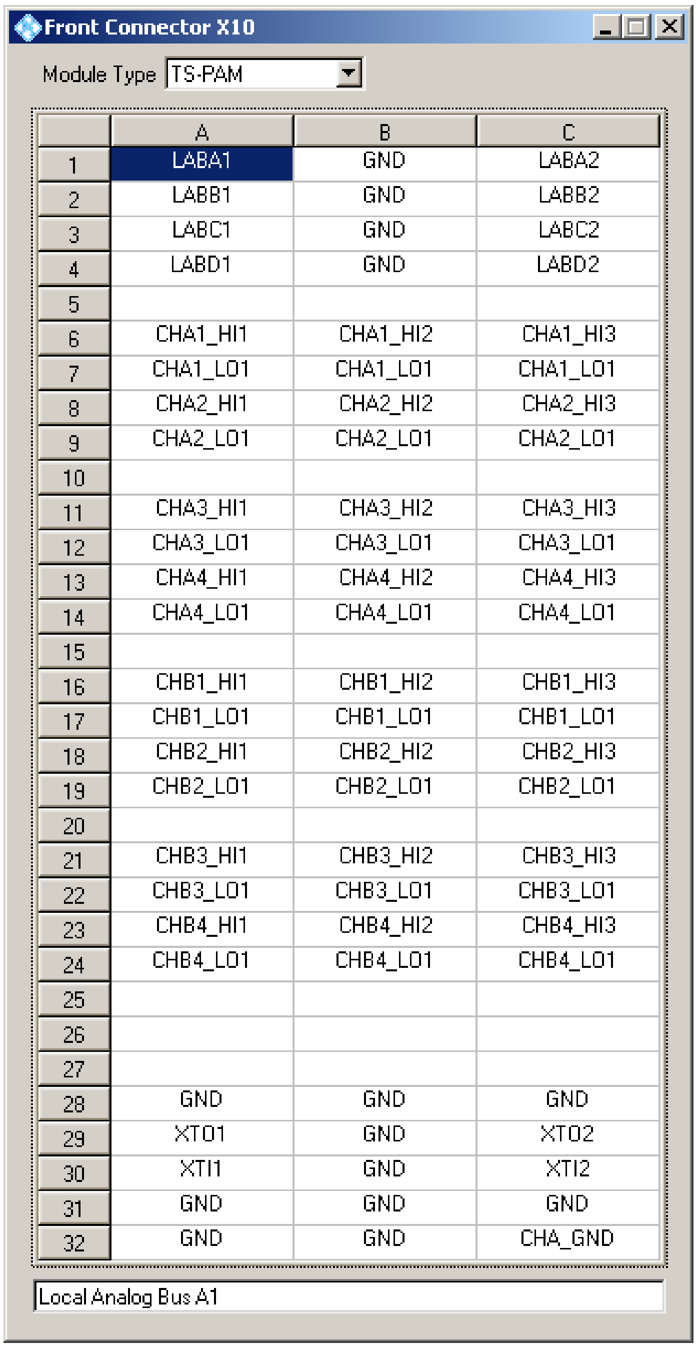
Overwrite the standard file `C:\Program Files\Rohde&Schwarz\Configuration\physical.ini` only if you are positive that you have not made any important changes or additions!

The following is an excerpt from an automatically generated `Physical.ini` file:

```
;; Created by tsvp_panel Version 01.12
;
[device->PSAM]
Description  = 'TS-PSAM Module 1'
Type         = PSAM
ResourceDesc = PXI1::10::INSTR
DriverDll    = rspsam.dll
DriverPrefix = rspsam
DriverOption = 'Simulate=0,RangeCheck=1'
; Note: the self test DLL and prefix keywords must be removed for the first
;       TS-PSAM module, because it is already tested in the basic self test.
; SFTDll     = sftmpsam.dll
; SFTPrefix  = SFTMPSAM

. . . . . . . . .

[device->PMB_10]
Description  = 'TS-PMB Module in Frame 1 Slot 10'
Type         = PMB
ResourceDesc = CAN0::0::1::10
DriverDll    = rspmb.dll
DriverPrefix = rspmb
DriverOption = 'Simulate=0,RangeCheck=1'
SFTDll       = sftmpmb.dll
SFTPrefix    = SFTMPMB

[device->PMB_14]
Description  = 'TS-PMB Module in Frame 1 Slot 14'
Type         = PMB
ResourceDesc = CAN0::0::1::14
DriverDll    = rspmb.dll
DriverPrefix = rspmb
DriverOption = 'Simulate=0,RangeCheck=1'
SFTDll       = sftmpmb.dll
SFTPrefix    = SFTMPMB

[device->PMB_15]
Description  = 'TS-PMB Module in Frame 1 Slot 15'
Type         = PMB
ResourceDesc = CAN0::0::1::15
DriverDll    = rspmb.dll
DriverPrefix = rspmb
DriverOption = 'Simulate=0,RangeCheck=1'
SFTDll       = sftmpmb.dll
SFTPrefix    = SFTMPMB

[device->PSYS1_15]
Description  = 'TS-PSYS1 Module in Frame 1 Slot 15'
Type         = PSYS1
ResourceDesc = CAN0::0::5::15
DriverDll    = rspsys.dll
DriverPrefix = rspsys
DriverOption = 'Simulate=0,RangeCheck=1'
SFTDll       = sftmpsys.dll
SFTPrefix    = SFTMPSYS

; The analog bus entry is mandatory if the GTSL Switch Manager or  EGTSL is used
[device->ABUS]
Type         = AB
```

13.4.3 Front Connectors

Using the **<Tools><Front Connectors>** menu item, you can display the front connector X10 pin assignment for a module. The front connector panel is also accessible using the **<Utility><Display Front Connector>** menu item from the instrument panels, or by pressing the **<F10>** key.



Module Type TS-PAM

	A	B	C
1	LABA1	GND	LABA2
2	LABB1	GND	LABB2
3	LABC1	GND	LABC2
4	LABD1	GND	LABD2
5			
6	CHA1_HI1	CHA1_HI2	CHA1_HI3
7	CHA1_LO1	CHA1_LO1	CHA1_LO1
8	CHA2_HI1	CHA2_HI2	CHA2_HI3
9	CHA2_LO1	CHA2_LO1	CHA2_LO1
10			
11	CHA3_HI1	CHA3_HI2	CHA3_HI3
12	CHA3_LO1	CHA3_LO1	CHA3_LO1
13	CHA4_HI1	CHA4_HI2	CHA4_HI3
14	CHA4_LO1	CHA4_LO1	CHA4_LO1
15			
16	CHB1_HI1	CHB1_HI2	CHB1_HI3
17	CHB1_LO1	CHB1_LO1	CHB1_LO1
18	CHB2_HI1	CHB2_HI2	CHB2_HI3
19	CHB2_LO1	CHB2_LO1	CHB2_LO1
20			
21	CHB3_HI1	CHB3_HI2	CHB3_HI3
22	CHB3_LO1	CHB3_LO1	CHB3_LO1
23	CHB4_HI1	CHB4_HI2	CHB4_HI3
24	CHB4_LO1	CHB4_LO1	CHB4_LO1
25			
26			
27			
28	GND	GND	GND
29	XTO1	GND	XTO2
30	XTI1	GND	XTI2
31	GND	GND	GND
32	GND	GND	CHA_GND

Local Analog Bus A1

Figure 13-14 Front Connector X10