Application Report
SNAA139A – April 2012 – Revised April 2013

AN-2245 MSP430 Interface to DAC161P997 Code Library

ABSTRACT

This application report describes the MSP430 software to interface and use the TI DAC161P997 devices. The accompanying software contains a function library allowing quick prototyping of the DAC161P997 setup and control. The software provided in this library is a starting point for developers wanting to get the most out of the MSP430 and the DAC161P997 devices.

Source code discussed in this application report can be downloaded from the DAC161P997 product folder.

Contents

List of Figures

List of Tables

1 Introduction

The MSP430 is an ideal microcontroller solution for low-cost, low-power precision sensor applications because it consumes very little power. The DAC161P997 is a 16- bit ΣΔ digital-to-analog converter (DAC) for transmitting an analog output current over an industry standard 4-20mA current loop. This library provides functions to facilitate the interfacing of any MSP430 device to a DAC161P997. Any device within the MSP430 family can be used with this library, made possible by hardware abstraction. This allows the designer maximum flexibility in choosing the best MSP430 device for the application. This document provides descriptive information and instructions for using the library either for demonstration purposes or implementation into a project. This is the recommended starting point for developing software for the DAC161P997 and MSP430 combination. The software examples have been developed for the MSP430F5528 breakout board, but can easily be ported to another hardware platform.

The DAC161P997 is a 16-bit ΣΔ digital-to-analog converter (DAC) for transmitting an analog output current over an industry standard 4-20mA current loop. The data link to the DAC161P997 is a Single Wire Interface (SWIF) that allows sensor data to be transferred in digital format over an isolation boundary. Error detection and handshaking features within the SWIF protocol ensure error free communication across the isolation boundary. The MSP430 is a great fit for applications where power conservation is a priority. The many power-saving mechanisms designed into the MSP430 make it ideal for such applications.

2 Purpose and Scope

To aid in interfacing these devices, TI has produced a code library that significantly reduces the need to write low-level interface functions. It provides a boost in the development of an MSP430/DAC161P997-based product, saving time and allowing quick progression to the application-specific aspects of the project. This library is designed to be used with any MSP430 device. Since a SWIF interface functions can be implemented using one of many peripherals within the MSP430 family, and since the peripherals available may differ by device and application, library calls are provided for each of these interfaces. The chosen interface is selected by assigning a value to a system variable, which causes the compiler to conditionally include the appropriate function calls. As such, application code utilizing the library remains portable between various MSP430 devices, with minimal modification required.

Two complete example application projects are provided with the library. The purpose is to demonstrate use of the library. It is not intended as a comprehensive guide to using the DAC161P997, and it does not make use of all the features of these devices. It does, however, use all the register access functions provided by the library.

3 File Organization

The library has been implemented with modular hardware abstraction. There is a header file specific to each of the hardware components (DAC161P997, MSP430, and the board). The hardware definition header files are shown in Table 1. Table 2 shows the library code files and its header, and Table 3 shows the demonstration applications that accompanies the library.

Table 1. Hardware Definition Files

Filename	Description
TI_DAC161P997.h	Definitions specific to the DAC161P997 device, including register locations and commonly-used masks for use with these registers. Symbol baud rate *BAUD_RATE* is chosen here. Permitted symbol baud rate range is 300-19200 symbols per second (sps). Symbol baud rate depends on the CPU clock defined by the macro definition *CPU_CLOCK*. This implementation uses 8MHz CPU clock.
TI_MSP430.h	Definitions specific to the MSP430 device; primarily, the pins used in the SWIF data link. Also, labels are defined for use with the system variable *TI_DAC161P997_SER_INTF*. This label selects the MSP430 peripheral (Timer_A0, Timer_B0, and GPIO bit bang) used to implement the *TI_DAC161P997_SWIFSendSymbol* function.
TI_MSP430_hardware_board.h	Definitions specific to the board being used such as the LED GPIO pin. SWIF connections are not defined here because they are defined within *TI_MSP430.h* The system variable *TI_DAC161P997_SER_INTF* is defined in this file.

Table 2. Library Code

Filename	Description
TI_MSP430_swif.c	Functions for accessing the DAC161P997 registers via SWIF from MSP430 family
TI_MSP430_swif.h	Function declarations for *TI_MSP430_swif.c*

Table 3. Demo Applications included with the Library

Demo Application	Filename	Description
Application1: Access SWIF to modify the DAC161P997 Registers	*demo-app01\main.c*	Application code with functions to unlock and modify the DAC161P997 Configuration Registers and then periodically update DACCODE register in a loop to output different currents
Application2: SWIF channel activity monitor	*demo-app02\main.c*	Application code to demonstrate SWIF channel activity monitor. Channel inactivity error detection is enabled and a timer interrupt service routine is used to send two valid symbols every 75ms to keep link active

Figure 1 shows a stack diagram of the library. Note that one of the files displayed in the stack is the standard definition file for the specific MSP430 device being used. This file is included with the development environment being used to create the MSP430 software.

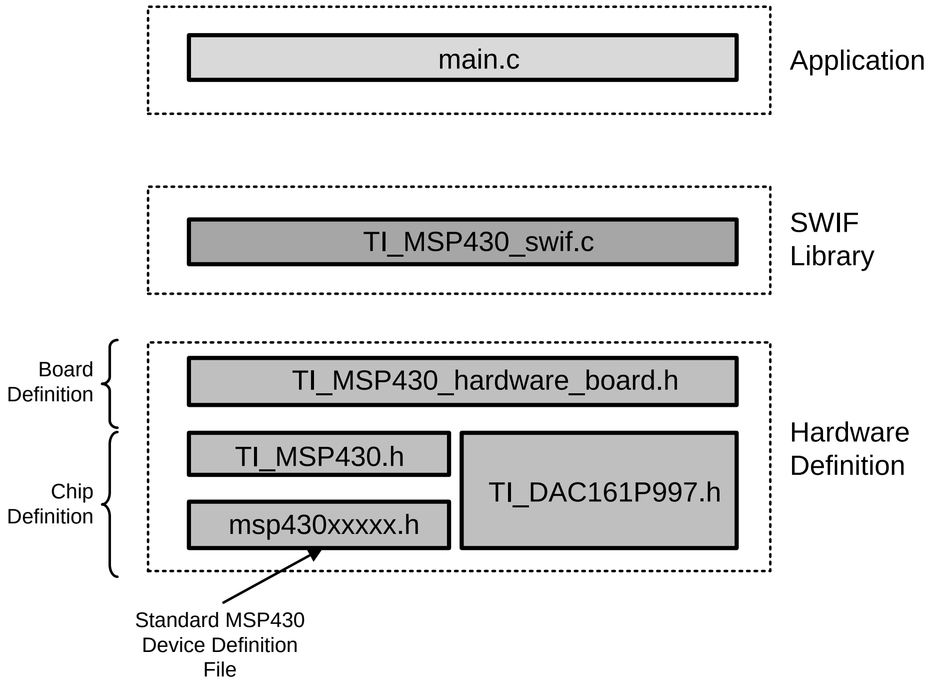

Figure 1. Code Library Stack

4 Functions

Table 4 shows the DAC161P997 register-access functions provided in the library, with a brief description.

Table 4. Register Access and Control Functions Provided by the Library

Function Name	Description
void TI_DAC161P997_SWIFSetup (void)	Configures the SWIF interface pins and must be called before calling any of the other functions.
int8_t TI_DAC161P997_SWIFWriteReg(uint16_t data, uint8_t tag)	Writes 16-bit value *"data"* to DACCODE register if *"tag"* is 0. If *"tag"* is 1, configuration register value is updated with *"data"*, in which case, the most significant byte of *"data"* indicates the configuration register and the least significant byte of *"data"* is the register value. Data bits are sent over SWIF as per the frame format described in the DAC161P997 data sheet. Following a valid frame, the SWIF is monitored for a receive acknowledgment symbol from the DAC161P997. If SWIF transfer is not successful (DAC161P997 acknowledge pulse not received or acknowledgement bit *ACK_EN* is not enabled), an error code *DAC_ACK_FAIL* is returned.
int8_t TI_DAC161P997_SWIFReceiveACK (void)	Checks for acknowledge pulse from the DAC161P997 on the SWIF receive pin. Returns an error code *DAC_ACK_FAIL* if the DAC161P997 fails to send an acknowledge pulse within the symbol period. This function is called by *TI_DAC161P997_SWIFWriteReg* to handle acknowledge from the DAC161P997.
void TI_DAC161P997_SWIFSendSymbol(uint8_t sym)	Sends symbol *"sym"* to the DAC161P997 on the SWIF transmit pin. Valid symbols, baud rate, and duty cycles are defined in *TI_DAC161P997.h*. This function is called by *TI_DAC161P997_SWIFWriteReg* to send a frame symbol to the DAC161P997.

4.1 *TI_DAC161P997_SWIF Setup*

For the SWIF data link implementation, the MSP430 uses three digital GPIO pins:

- One output pin *TI_DAC161P997_SWIF_PRITX_PIN* (referred as *PRI_TX* in the discussion below) for transmitting encoded data to the DAC161P997
- One input pin *TI_DAC161P997_SWIF_PRIRX_PIN* (referred as *PRI_RX* in the discussion below) for receiving the acknowledge signal from the DAC161P997
- One output pin *TI_DAC161P997_SWIF_PRITX_EN_N_PIN* (referred as *PRI_TX_EN_N* in discussion below) that governs the direction of the data flow over the SWIF link

For details of the SWIF implementation, see the DAC161P997EVAL user's guide [3] and the DAC161P997 data sheet [1].

The MSP430 GPIO pin assignments for the DAC161P997 SWIF data link are defined in the *TI_MSP430.h* and the *TI_DAC161P997_SWIFSetup* function and called to configure these pins. This function must be called before any of the other functions.

4.2 *TI_DAC161P997_SWIFWriteReg*

The *TI_DAC161P997_SWIFWriteReg* function accepts the 16-bit payload and tag bit as inputs and automatically generates the data frame with parity bits for transfer over the SWIF. For details of the SWIF data frame, see the DAC161P997 data sheet [1]. The *TI_DAC161P997_SWIFWriteReg* function writes a byte of data into the specified configuration register address if *"tag"* sent is 1. Alternatively, it can write a 16-bit value to the DACCODE register if *"tag"* sent is 0. It calls the function *TI_DAC161P997_SWIFSendSymbol* to send frame symbols over the SWIF *PRI_TX* pin using duty cycle encoded waveform. The *TI_DAC161P997_SWIFReceiveACK* function is called at the end to handle the acknowledge symbol from the DAC161P997. If SWIF transfer is not successful (DAC161P997 acknowledge pulse not received or acknowledgement bit ACK_EN is not enabled), an error code *DAC_ACK_FAIL* is returned.

4.3 *TI_DAC161P997_SWIFSendSymbol*

This function is used to send data symbols of the frame over SWIF *PRI_TX* pin as duty cycle encoded waveform. The individual symbol period is defined through the *BAUD_RATE* and *CPU_CLOCK* macro definitions in *TI_DAC161P997.h*. The baud rate valid range is 300 to 19200 symbols per second. A version of the *TI_DAC161P997_SWIFSendSymbol* function is provided for the following MSP430 peripherals:

- Timer_A0
- Timer_B0
- GPIO BitBang (default)

The system variable *TI_DAC161P997_SER_INTF* selects the peripheral to be used for implementing the *TI_DAC161P997_SWIFSendSymbol* library function.

5 Using the Software

5.1 *Prerequisites*

To successfully compile, download and run the software described in this document, the following materials are needed:

- MSP430 Target Board MSP-TS430RGC64USB Board
- DAC161P997 Evaluation Board DAC161P997EVAL
- MSP430 USB Debugging Interface MSP430-FET430UIF
- IAR Embedded Workbench or TI Code Composer Studio™ for MSP430

The software can be adapted to run on other MSP430 hardware boards as well. For instructions, see Section 4.3.

A free, code size limited, but fully functional edition of IAR Embedded Workbench (IAR Kickstart) is available from the IAR Systems website *(*www.iar.com) or from the TI MSP430 software tools page:

http://www.ti.com/lsds/ti/microcontroller/16-bit_msp430/msp430_software_landing.page.

As an alternative to IAR Embedded Workbench, it would be possible to use Code Composer. A trial edition of the Code Composer is available from the TI MSP430 software tools page.

5.2 *Getting Started*

Follow these simple steps to get your application up and running:

1. Install IAR Workbench.
2. Download the source code for this application report and unzip the files to your working directory.
3. Open IAR Embedded Workbench and create a new project:
   (a) Select Project → Create new project.
   (b) Select tool chain MSP430.
   (c) Base the project on the empty project template.
   (d) Save (you will also be asked to save the current workspace).
4. Add the following C files from the software that you downloaded and unzipped in step 2:
   (a) All C files from the *code\library* folder.
   (b) All C files from the *code\demo-application-examples\demo-app01* folder.

5. Open the "options..." dialog for the new project by right clicking the project name in the workspace window. A window should appear.
   (a) Under "General options", select the MSP device. For the MSP-TS430RGC64USB target board, use MSP430F5528.
   (b) Under "C/C++ compiler", click on the preprocessor pane.
      (i) The "Ignore standard include directories" tick box should not be ticked.
      (ii) In the "Additional include directories", add include paths telling the compiler where to find the header files included by the C files. You should add the *$PROJ_DIR$\code\include* folder.
   (c) Under "Debugger", select "FET Debugger" from the "Driver" drop down list.
   (d) Under "FET Debugger", in the "Connection" section, choose the connection type of the FET tool (for example, Texas Instruments USB-IF). Leave the rest of the settings as is.
6. Click “OK” to close the options window.
7. Select “Project → Rebuild All”. There should be no errors or warnings when IAR rebuilds the executables (if not done already, you will also be asked to save the current workspace).
8. The configuration of the hardware definition files in the library as distributed by TI is for an MSP430F5528 equipped board. The system variable *TI_DAC161P997_SER_INTF* defined within *TI_hardware_board.h* identifies *BITBANG* as the method to implement the DAC161P997 SWIF functions. *TI_MSP430.h* identifies the pins used for SWIF pins implementation.
9. Connect the DAC161P997 evaluation board SWIF interface lines to the MSP430 target board pins as shown in Figure 2.

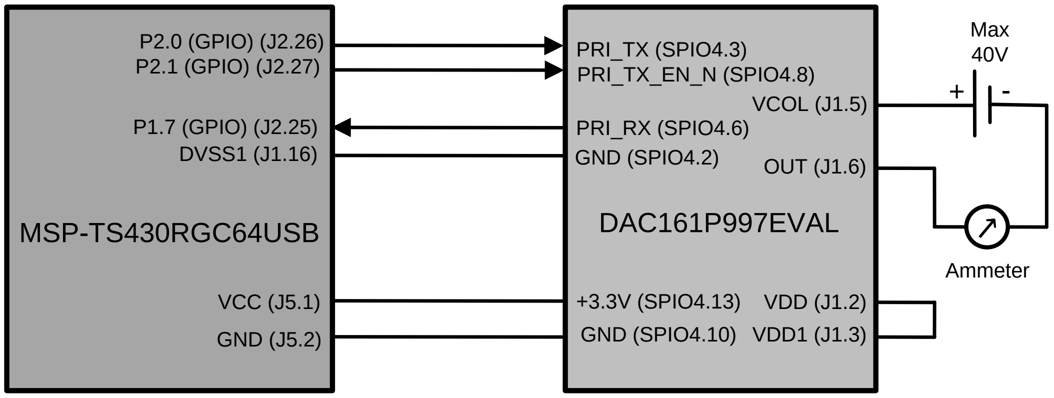

Figure 2. DAC161P997 to MSP430 Connection Diagram

10. The DAC161P997 evaluation board jumper connections can be seen in Figure 3 and Table 5. The 4-20mA loop setup connections should be made between terminals 5 and 6 of J1 connector as described in the DAC161P997 user guide [3].

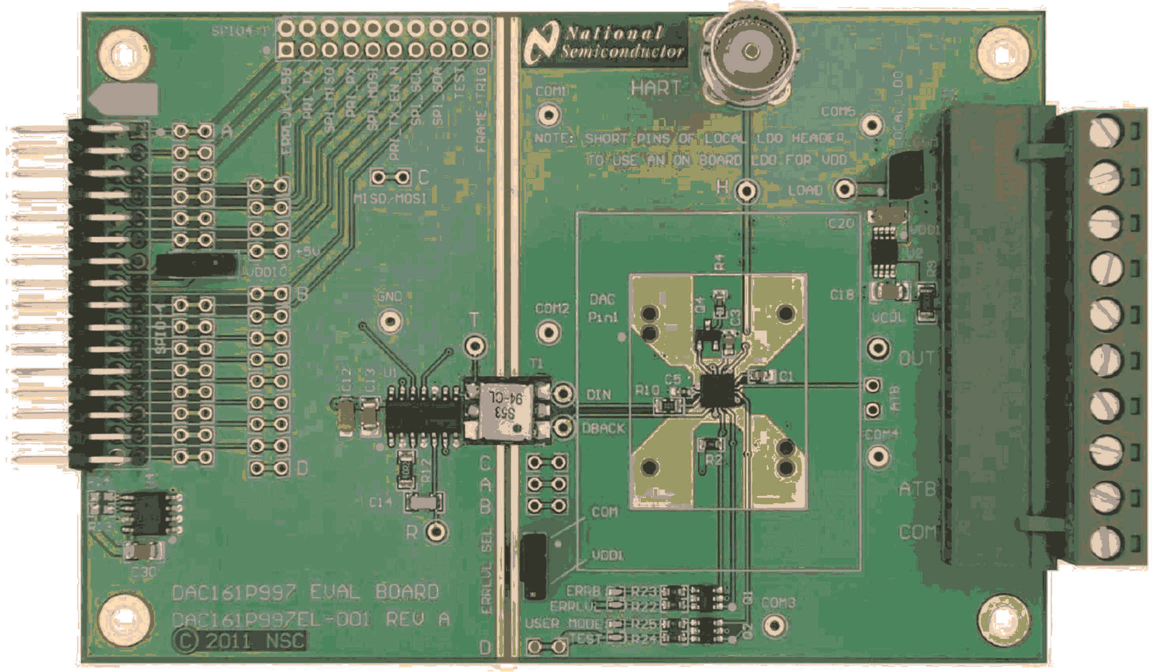

Figure 3. DAC161P997SDEVAL Jumper Settings

Table 5. DAC161P997EVAL Jumper Settings (Default Settings)

Jumpers	Pin	Purpose
VDDIO	P1-P2	+3.3V supply for onboard buffer
ERRLVL_SEL	P1-P2	Output current level selection for error condition
LOCAL_LDO	P1-P2	Generate the DAC161P997 supply from loop voltage

11. Attach the MSP430 FET to your PC. If you are running Windows and using the USB FET tool for the first time, you will asked to install some drivers for the tool. For Windows they are located in *$IAR_INSTALL_DIR$\430\drivers\TIUSBFET*.
12. Attach the MSP430 FET to the MSP430 target board using the JTAG connector. The VCC power select jumper JP3 should be set to 1-2 (int) where the board is powered from the FET alone. The MSP430 target board jumper connections can be seen in Figure 4.

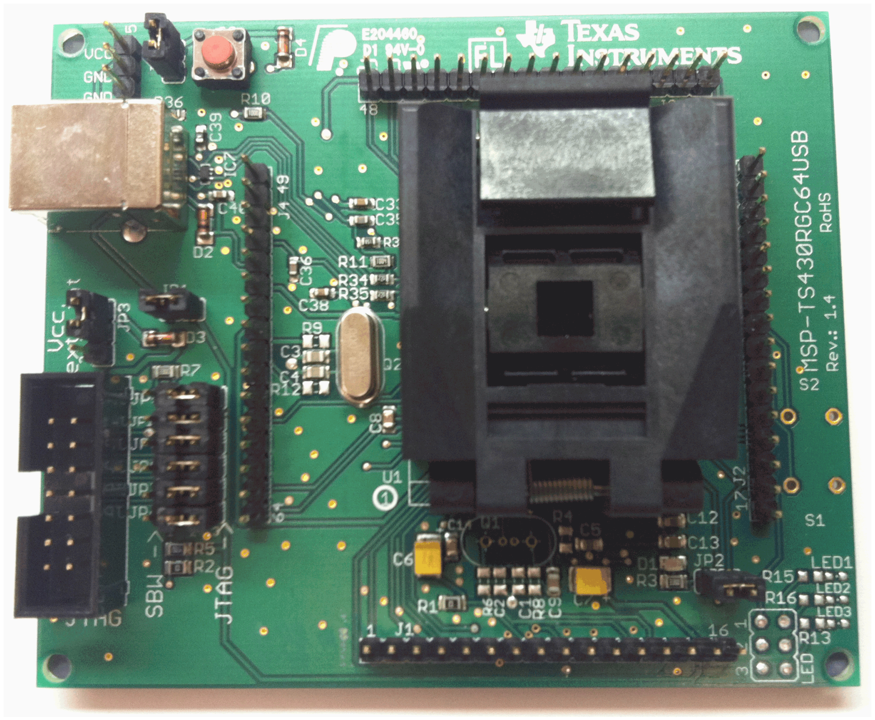

Figure 4. MSP-TS430RGC64USB Jumper Settings

13. Select Project → Debug. IAR will now establish a connection with the target MCU, download the application and program the MSP430. The debugger will be started, halting the target at *main ()*.
14. Demo_app01 is a simple example that demonstrates the use of library functions to unlock and modify the DAC161P997 configuration registers. It then periodically updates the DACCODE register in a loop to output different currents. The onboard LED on the MSP430 target board is setup to continuously blink between different current updates.
15. Steps 3 to 14 can be followed to exercise other demo applications included with the library as well.

5.3 Adapting the Demo Project to Other Hardware

The procedure for adapting this code to other hardware is as follows:

1. Edit the pin assignments within *TI_MSP430.h* for the DAC161P997 SWIF interface pins. The labels being referenced in the *#define* assignments will be drawn from the standard definition file *(msp430.h)* listed at the top of *TI_MSP430.h*.
2. Edit the pin assignments in *TI_MSP430_hardware_board.h*, taking into account all the necessary connections on the board being used. The assigned labels are drawn from the standard definition file *(msp430.h)* listed at the top of *TI_MSP430.h*.
3. Assign the proper values to *TI_DAC161P997_SER_INTF* in *TI_MSP430_hardware_board.h*. The labels available for assignment can be found at the bottom of *TI_MSP430.h*.
4. Set up appropriately the function (*usc_init()*) to configure the system clock source and clock rate. This depends on your hardware and the particular MSP430 MCU in use. Based on this, setup valid *BAUD_RATE* and *CPU_CLK* macro definitions in *TI_DAC161P997.h.*. Make sure the physical hardware connections between the MSP430 target board and the DAC161P997EVAL are modified according to the pin assignments above. After making these changes, rebuild the project and download the code image. The application should function as described earlier.

5.4 *Using the Library With an Application*

The same procedure as described in the section above should be applied in order to adapt the library to the new hardware.

The function *TI_DAC161P997_SWIFSetup()* should always be called after a POR event within the MSP430. After this the access of registers is straightforward.

6 References

1. *DAC161P997 Single-Wire 16-bit DAC for 4-20mA Loops Data Sheet* (SNAS515)
2. DAC161P997 Evaluation Development Platform Software
3. *DAC161P997 Single-Wire 16-bit DAC for 4-20mA Loops Evaluation Board User’s Guide* (SNAU073)
4. *MSP430F551x and MSP430F552x Mixed Signal Microcontroller Data Sheet* (SLAS590)
5. *MSP430x5xx and MSP430x6xx Family User’s Guide* (SLAU208)
6. *MSP430 Hardware Tools User's Guide User's Guide* (TS430RGC64USB) (SLAU278)
7. *MSP430 Hardware Tools User's Guide User's Guide* (MSP-FET430UIF) (SLAU278)

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		