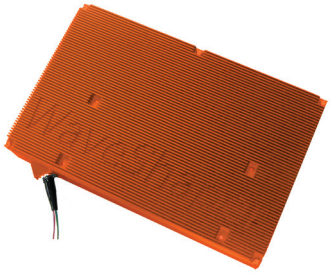


WaveShaper User Manual

FINISAR®

Part Number 1134082
Rev J00

Important note: This guide covers operation of the WaveShaper S Series and M Series hardware with the following software packages:

- **WaveManager Application Suite 2.7**
- **WSUTIL 1.5**
- **WaveShaper API 1.5**

It is not valid for other hardware or software releases.

FINISAR

Finisar Corporation
1389 Moffett Park Drive
Sunnyvale, CA 94089-1133
USA

(US Toll Free): 888.746.6484
Phone: +1 408.400.1110

www.finisar.com

Contents

Section 1: Manual Overview **1**

1.1 Introduction 1

1.2 About this manual 2

Section 2: Getting Started **3**

2.1 Safety Considerations - WaveShaper S-series 3

2.2 Safety Considerations - WaveShaper M-series 4

2.3 Laser Safety 5

2.4 Unpacking and Inspection 5

2.5 WaveShaper General Specifications 6

2.6 Control Computer Requirements 6

2.7 Windows Installation 6

2.8 Linux Installation 10

2.9 Connecting the WaveShaper to other equipment 10

2.10 Start-up and Initialization 10

Section 3: WaveShaper Overview **13**

3.1 LCoS Overview 13

3.2 Optical Design 14

3.3 Operating Conditions 14

3.4 Calibration and Maintenance 15

3.5 Storage 15

3.6 WaveShaper Optical Specification 15

Section 4: WaveManager Software Overview **17**

4.1 User Interface Overview 17

4.2 Explorer Panel 21

4.3 Adding a WaveShaper during program operation 22

4.4 Removing a WaveShaper during program operation 23

4.5 Multiple configuration file for 16000 units: 23

4.6 Options 24

Section 5: Controlling the WaveShaper Spectrum with the WaveManager Application Suite **25**

5.1 Overview 25

5.2 Basic sub-tab 26

5.3 Advanced sub-tab 28

5.4 Preset sub-tab 29

5.5 Flexgrid sub-tab 30

Section 6: WaveShaper File Formats 35

6.1 Overview 35

6.2 User Configured Filters (**.ucf*) files 35

6.3 WaveShaper Preset Files (**.wsp*) files 38

6.4 flexgrid (**.wsgrid*) files 39

6.5 WaveShaper Preset Power Split Profile (**.psp*) files 40

Section 7: WaveShaper WSUTIL User Guide 43

7.1 Introduction 43

7.2 Installation 43

7.3 User Instructions 43

Section 8: WaveShaper API Programming Guide 47

8.1 Overview 47

8.2 Development Environment Setup 47

8.3 WS Command Overview and Program Structure 49

8.4 WS API Functions 54

8.5 PS Command Overview and Program Structure 67

8.6 PS API Functions 69

8.7 Error Code Type Definitions 73

Section 9: WaveShaper Specifications 75

9.1 WaveShaper M Series General Specifications 75

9.2 WaveShaper S Series General Specifications 78

9.3 Optical Specifications 81

1.1 Introduction

1.2 About this manual

Section 1: Manual Overview

1.1 Introduction

The WaveShaper family of programmable optical processors and filters provide extremely fine control of filter amplitude and phase characteristics across the C-Band, L-Band or C+L bands. Based on Finisar's high-resolution, solid-state Liquid Crystal on Silicon (LCoS) optical engine, the optical transfer function can be software-specified by the user across the entire 5 THz of operation (9 THz for C+L) , which allows the generation of arbitrary, complex filters as well as control of filter bandwidth and centre frequency with 1 GHz resolution. They contain no moving parts and so provide extremely high reliability for both laboratory and production test applications.

The WaveShaper 100 Channel Selector provides precise control of filter centre frequency and bandwidth, selectable built-in Flat-top and Gaussian filter shapes, whilst the WaveShaper 120 is optimised for use as a high-precision gain-flattening element for advanced EDFA designs. The WaveShaper 1000 enables the additional capability of attenuation control and precise control of filter shape and phase with programmable predefined and user-generated filter shapes.

The WaveShaper 4000 adds 4-port switching functionality, making it possible to switch or combine multiple signals at different wavelengths in an "Add" or "Drop" configuration. In addition the WaveShaper 4000 has the ability to emulate the Flexgrid™ capability of a Finisar Wavelength Selective Switch which allows the user to prototype advanced optical networks for 400 Gbit/s and higher data rates. The WaveShaper 4000 also has the ability to direct light into up to 4 ports simultaneously at a given frequency. The power levels at each port can be adjusted arbitrarily. This then transforms the WaveShaper into a versatile optical device emulator. It can be used to emulate Mach Zhender interferometers, 1x4 couplers with arbitrary coupling ratios, or DQPSK/OFDM demux filters.

The WaveShaper 16000 provides the same switchng and power-splitting capabilities as the WaveShaper 4000, but with 16 output ports. It also has the ability to operate as an MxN switch to emulate the functionality of an MxN WSS

The WaveShaper family are available in benchtop and rack-mounted versions (S-Series) as well as in OEM versions for embedding into third party equipment and instrumentation (M-Series). All WaveShapers share a common interface and can be controlled either through a graphical user interface (WaveManager) or through a fully-documented API. Both Windows and Linux operating systems are supported, which along with full diagnostic and monitoring capabilities, makes it easy to develop and integrate code into the user's system.

WaveSketch is a complementary tool to the WaveManager Suite of applications available under the supported Windows operating systems. It is an interactive graphic tool which modifies the filter profile of a WaveShaper 120, 1000 or 4000 in real time. It

allows the user to draw arbitrary filter shapes which get automatically uploaded to the WaveShaper.

The WaveShaper Fourier Processor Application is an addition to the WaveManager Suite of applications for the WaveShaper 4000 and 16000 families available under the supported Windows operating systems. It provides tools for rapid prototyping optical circuits, enabling functions such as power sharing at each wavelength across multiple ports with attenuation and phase control to impose different delays on the different signal parts. This allows the user to program arbitrary types of single-input, multi-output interferometers, e.g. Mach-Zehnder Interferometers (MZI) , Differential Quadrature Phase-Shift Keying DPSK demodulators, etc. Other applications could be crosstalk emulation/suppression and broadcasting

1.2 About this manual

This manual is designed for all models of the WaveShaper family and covers the WaveManager Software Suite v2.6.4 and above, WSUTIL and the WaveShaper programming interface for controlling the WaveShaper.

The complementary WaveSketch Application and Fourier Processing Application each have their own built in manuals available via the application "help" menu.

Section 2 describes how to set up and install the WaveShaper and its associated control software. It also caters for optional installation of the the complementary WaveSketch and Fourier Processor Applications. **It is critical that this section is read and understood as it contains important safety and operating instructions which, if not followed, could result in personal injury or damage to the WaveShaper unit.**

Section 3 describes how the WaveShaper works, whilst Section 4 provides an introduction to the WaveManager control software, a Java-based graphical user interface, which is described in detail in Section 5.

One of the unique capabilities of the WaveShaper is the ability for the user to generate and load arbitrary user-defined optical transfer functions (filter shapes.) To enable this, there a number of file formats supported by the WaveManager and Fourier Processor software and these are explained in Section 6

In addition to the WaveManager control software, a simple command-line interface - WSUTIL - is provided for computing and loading filter profiles on to a WaveShaper device. Operation of this utility is described in Section 7.

The WaveShaper family also supports comprehensive Application Programming Interface (API). Section 8 is provided for application developers who wish to control a WaveShaper using the API command set. It includes a list of all the command functions together with coding examples in C. Some sample code is included for both C and Python.

LabVIEW support for the WaveShaper is available in a separate document, available from the WaveShaper downloads at http://www.finisar.com/WaveShaper or on request.

Specifications for all members of the WaveShaper family are given in Section 9.

2.1 Safety Considerations - WaveShaper S-series

2.2 Safety Considerations - WaveShaper M-series

2.3 Laser Safety

2.4 Unpacking and Inspection

2.5 WaveShaper General Specifications

2.6 Control Computer Requirements

2.7 Windows Installation

2.8 Linux Installation

2.9 Connecting the WaveShaper to other equipment

2.10 Start-up and Initialization

Section 2: Getting Started

2.1 Safety Considerations - WaveShaper S-series

The following general safety precautions must be observed during all phases of operation, service, and repair of this module. Failure to comply with these precautions or with specific warnings elsewhere in this manual violates safety standards of design, manufacture, and intended use of the instrument. Finisar assumes no liability for the customer's failure to comply with these requirements.

This product has been designed in accordance with IEC Publication 61010-1, Safety Requirements for Electrical Equipment for Measurement, Control and Laboratory Use, and has been supplied in a safe condition. The instruction documentation contains information and warnings that must be followed by the user to ensure safe operation and to maintain the product in a safe condition.

Before operation, you should review the instrument and manual for safety markings and instructions. You must follow these to ensure safe operation and to maintain the instrument in safe condition.

Warning

To avoid hazardous electrical shock, do not perform electrical tests when there are signs of shipping damage to any portion of the outer enclosure.

Line Power

The WaveShaper "S" family of Programmable Optical Processors are designed to operate from a single-phase AC power source that supplies between 100 and 240 V at frequencies in the range 50 - 60 Hz. The maximum power consumption is 50 VA.

In accordance with international safety standards, the instrument has a three-wire power cable. When connected to an appropriate AC power receptacle, this cable earths the instrument chassis.

Warning

To avoid the possibility of injury or death, you must observe the following precautions before switching on the instrument.

- **Insert the power cable plug only into a socket outlet provided with a protective earth contact. Do not negate this protective action by using an extension cord or power block without a protective conductor.**
- **Do not interrupt the protective earth connection intentionally.**
- **Do not remove protective covers. Operating personnel must not remove instrument covers. The unit contains no user-serviceable components. Component replacement and internal adjustments must be made only by qualified Finisar service personnel.**
- **A WaveShaper chassis that appears damaged or defective should be made inoperative and secured against unintended operation until it can be repaired by qualified Finisar service personnel.**
- **A defective, damaged, or malfunctioning WaveShaper chassis must be returned to Finisar for repair**.

2.2 Safety Considerations - WaveShaper M-series

The OEM versions of WaveShaper (M-Series) of Programmable Optical Processors are designed to operate from a user-supplied +5V supply.

Warning

- **The module contains Electrostatic Discharge Sensitive Devices.**
- **Operating personnel must not remove the module lid. The module contains no user-serviceable components. Component replacement and internal adjustments must be made only by qualified Finisar service personnel.**
- **A WaveShaper module that appears damaged or defective should be made inoperative and secured against unintended operation until it can be repaired by qualified Finisar service personnel.**
- **A defective, damaged, or malfunctioning WaveShaper module must be returned to Finisar for repair**.

The WaveShaper M-Series Module is supplied with a mating Phoenix power plug. This plug needs to be wired to a power supply capable of providing +5 V to +5.6 V capable of delivering a current of at least 8 A.

The wire used should be red for +5 V and black for 0 V 22 AWG (0.5 mm,16/0.2,Tri-rated PVC insulated UL1015).

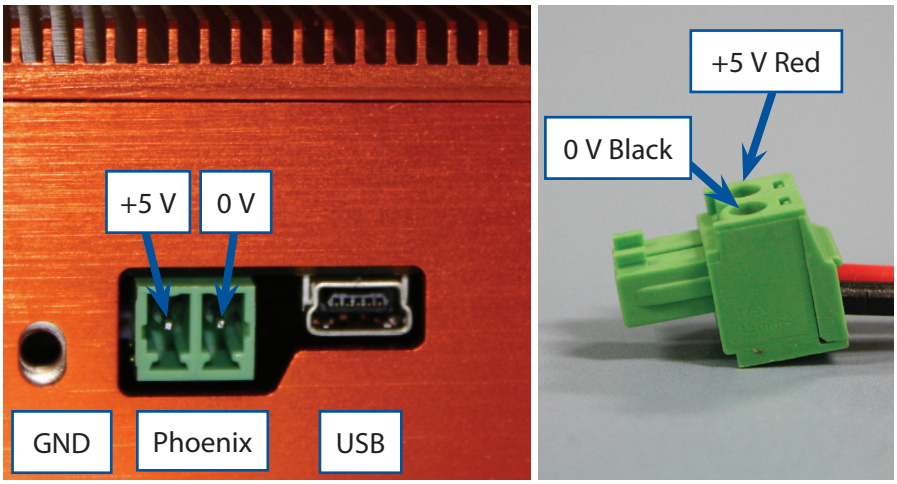


Figure 2.1 Phoenix Connector Power Wiring

2.3 Laser Safety

The WaveShaper family of Programmable Optical Processors are designed for use with various classes of laser up to, and including, Class 3B lasers. Whilst the WaveShaper module does not generate laser light, laser light may be present on one or more output ports depending on the configuration of WaveShaper selected and the type of laser connected to the input port(s).

Warning

Please pay attention to the following laser safety warnings:

- **Under no circumstances look into the end of an optical output cable/connector(s) when the device is operational. If there is any laser radiation it could seriously damage your eyesight.**
- **Do not operate the WaveShaper without attaching the optical output connector(s) to a safely terminated mating connector(s).**
- **Refer servicing only to qualified and authorized Finisar personnel.**

2.4 Unpacking and Inspection

On receipt, please check the unit for any signs of damage that may have been caused in transit, and check that the inventory is complete. Report any damage or missing parts to your shipping agent or Finisar representative.

The WaveShaper packaging contains the following components:

WaveShaper S-Series	WaveShaper M-Series
WaveShaper Unit	OEM WaveShaper Module
Approved Mains Power Plugs	Phoenix connector for supplying module power
USB 2.0 Type A male to Type B male-cable	USB 2.0 mini Type B male to Type A male cable
WaveManager Installation CD or USB Drive	WaveManager Installation CD or USB Drive
User Manual (This manual)	User Manual (This manual)

2.5 WaveShaper General Specifications

To ensure correct and safe operation of the WaveShaper, the appropriate WaveShaper specifications should be adhered to. These specifications can be found in the Specification section at the rear of the manual. Please refer to your serial number label for the correct model and part number

2.6 Control Computer Requirements

The computer used to control the WaveShaper using Finisar's WaveManager Application Suite has the following minimum hardware and software requirements:

- 150 MB of hard disk space for a complete installation (including Java Runtime)
- 1 GB of RAM
- USB 2.0 port
- Display resolution of WSVGA (1024 x 600) or higher
- Microsoft Windows or a suitable version of Linux as defined in Table 2.1

OS	OS Release Supported
Windows	Windows 7, 32 bit and 64 bit version
	Windows 8.1, 32 bit and 64 bit version
	Windows 10, 32 and 64 bit version
Linux	2.6 Kernel or higher

Table 2.1 Operating Systems supported by WaveManager Application Suite Release 2.7 and above

The WaveManager Application Suite requires Java Runtime version 6 or higher. This is provided as part of the installation package and is automatically installed during the installation process.

Note that the WaveManager Application Suite will run on Wndows XP SP3 but is not supported on that platform. Windows 7 or above is recommended for stable operation

2.7 Windows Installation

! **For trouble-free operation of the WaveShaper, it is essential that the WaveManager Application Suite software is installed BEFORE connecting the WaveShaper to the computer to be used to control the WaveShaper. Please ensure that the software installation is completed before connecting the WaveShaper to the PC .**

! **Please ensure that all appropriate Service Packs have been installed prior to installing the WaveShaper software.**

! **The installation requires Administrator rights on the target computer, but subsequent operation will require only standard user rights**

To install the WaveManager Application Suite, start the *Setup.exe* file which can be found on the Installation CD or USB Drive and follow the installation procedure as out-

lined below. The installation process will automatically install all necessary files.

1. Please make sure any WaveShapers are disconnected from the computer, then click Next at the welcome screen (Figure 2.2).

Figure 2.2 WaveManager Installer welcome screen

2. Review the license agreements, and click "I Agree" (Figure 2.3) to proceed.

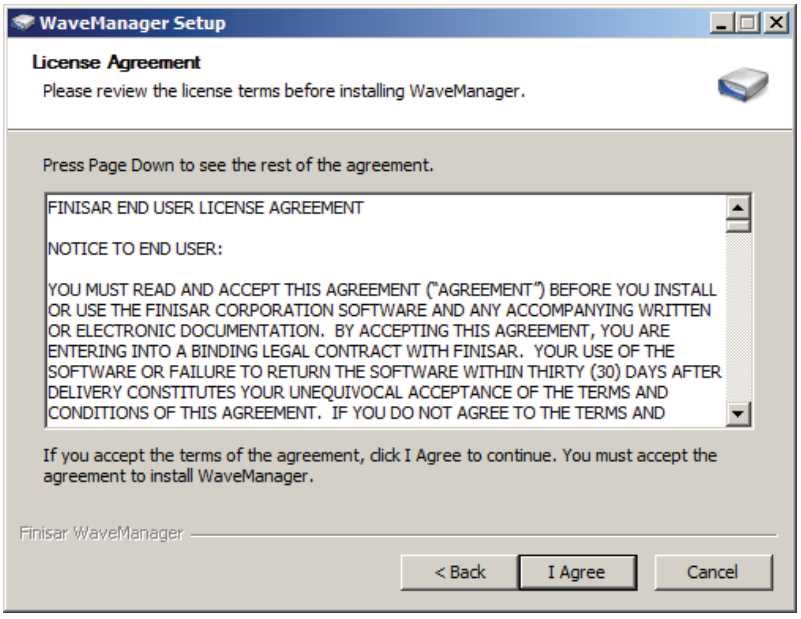


Figure 2.3 License agreement screen

3. At the Destination Directory screen (Figure 2.4) make any changes required to the file locations then click Next.

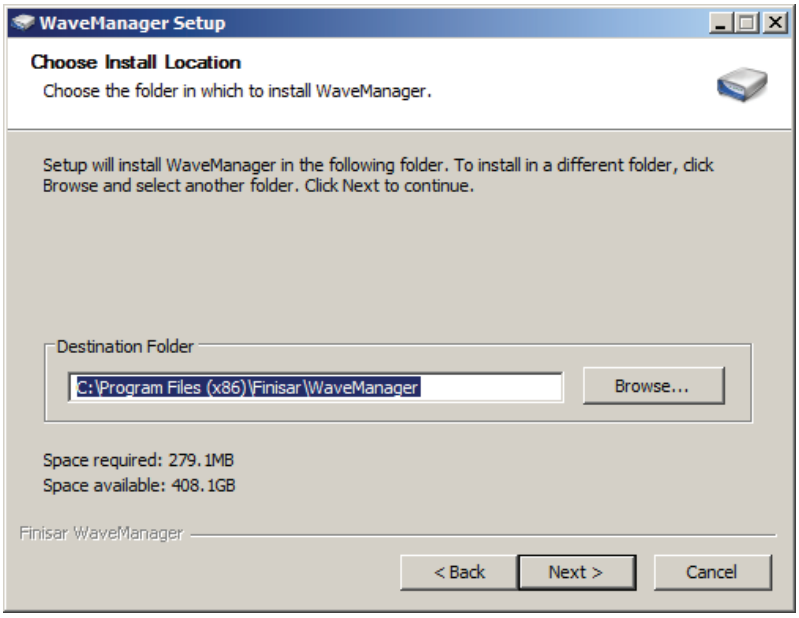


Figure 2.4 Enter name of directory for WaveManager installation

4. De-Select any optional components you do not wish to be installed then click <Next> (Figure 2.5).

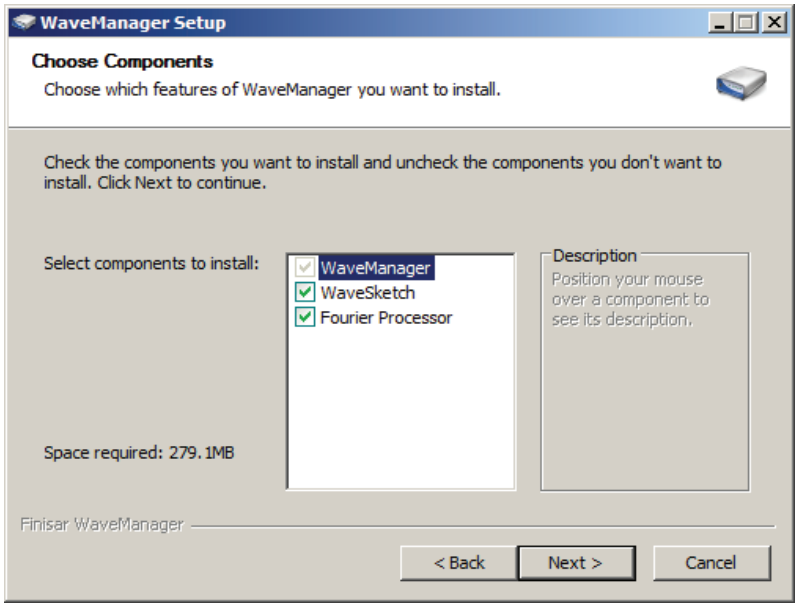


Figure 2.5 Enter name of Start folder

5. Select the name of the Start Menu folder, then click <Install> (Figure 2.6).

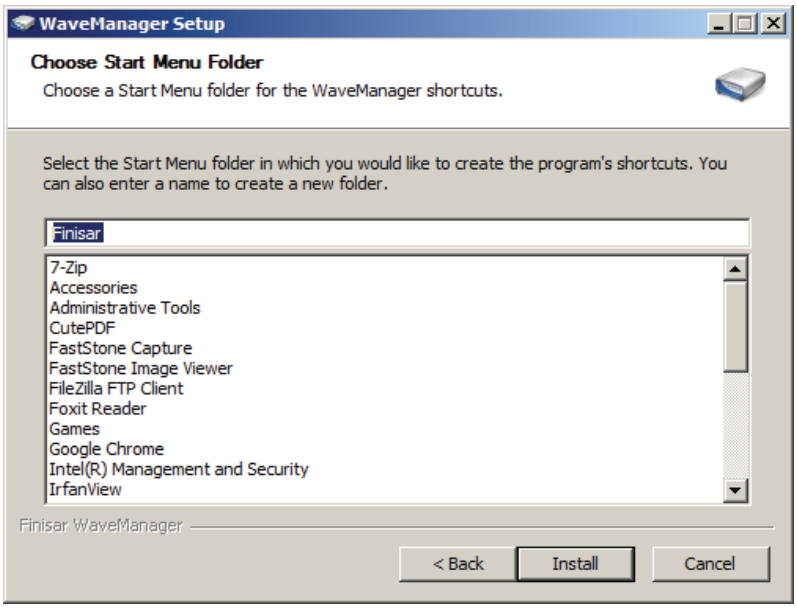


Figure 2.6 Enter name of Start folder

6. The files will now be installed (Figure 2.7)

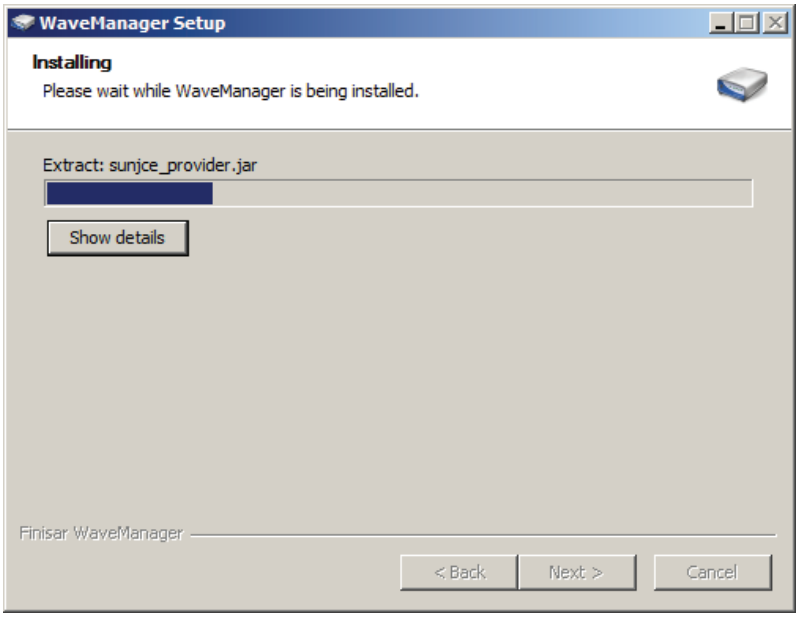


Figure 2.7 WaveManager Installation progress

7. Click "Finish" to complete the installation process (Figure 2.8)

Figure 2.8 WaveManager Installation Wizard Completion Screen

Note: Hardware Drivers

A WaveShaper is a 'Compound Device' in USB-speak and as such contains a number of USB-based hardware interfaces connected through a common USB hub. Depending on the configuration of the computer, additional drivers may need to be installed to manage these interfaces. These drivers are stored in the *\drivers* directory on the installation CD or USB Drive supplied. If these additional drivers are required, Windows will attempt to install them and may request a location where the drivers can be found. In case Windows cannot find the drivers automatically, please run the installer found in the *\drivers* directory on the installation CD or USB Drive prior to connecting the WaveShaper to the computer, so that Windows will automatically find the necessary drivers already installed on the system.

2.8 Linux Installation

For Linux platforms, the package is provided as a *.tar.gz* file that needs to be uncompressed in an appropriate directory. The user is responsible for making sure the correct drivers and Java runtime environment are installed and accessible. Linux kernel builds of 2.6 or higher normally contain all necessary drivers by default.

2.9 Connecting the WaveShaper to other equipment

The WaveShaper has optical connectors for input and output. Care should be taken with optical interface cleanliness. Connector ends should be cleaned using isopropyl alcohol and a lint-free tissue, or proprietary connector cleaning cassette, before mating.

2.10 Start-up and Initialization

To start the WaveShaper, connect power to the WaveShaper, turn on the WaveShaper and ensure the front panel LED is lit (WaveShaper S-series only). At start-up, the WaveShaper will initially be set to a blocked state, where the output optical power is minimized.

Ensure the WaveShaper is connected to the computer using a USB cable and then start the WaveManager Application Suite. It should be noted that the WaveShaper has a warm-up time of up to 10 minutes following start-up, during which time it will function correctly, but the performance is not guaranteed to meet all specifications.

When the WaveManager Application Suite is started it will first configure all necessary drivers and communication ports, then identify and connect to any Finisar WaveShaper devices attached to the computer. The process of identification and connection to a WaveShaper does not change the current settings of that device.

The WaveManager Display (see Section 4) will be populated with tabbed control panels for each WaveShaper identified and connected to. Should a WaveShaper device be identified but have no matching *wsconfig* file, a popup window will prompt the user to transfer the device *wsconfig* file to the appropriate directory on the computer. Should no devices be connected or identified, the explorer pane in the WaveManager Window allows for simulations to be run against existing *wsconfig* files.

Section 3: WaveShaper Overview

The WaveShaper family of Programmable Optical Processors provide unique capabilities for the manipulation and transformation of optical signals. They are based on Finisar's field-proven Liquid Crystal on Silicon (LCoS) switching element.

3.1 LCoS Overview

3.2 Optical Design

3.3 Operating Conditions

3.4 Calibration and Maintenance

3.5 Storage

3.6 WaveShaper Optical Specification

3.1 LCoS Overview

Conventional liquid crystal components used in telecom applications employ control of polarisation state to pass or transmit light to create for example wavelength blockers or attenuators. Switching applications can be achieved through polarisation dependent deflection or displacement. These designs are limited by having inflexible configurations – typically with one pixel per channel - and the requirement to pre-configure the channel plan in advance.

Figure 3.1 Schematic of LCoS structure

Liquid Crystal on Silicon (LCoS) is a display technology which combines Liquid Crystal and semiconductor technologies, to create a high resolution, solid-state display engine. Figure 3.1 shows the structure of an LCoS display with the Liquid Crystal (LC) layer sandwiched between the Active Matrix silicon backplane and the ITO-coated (Indium Titanium Oxide) top glass.

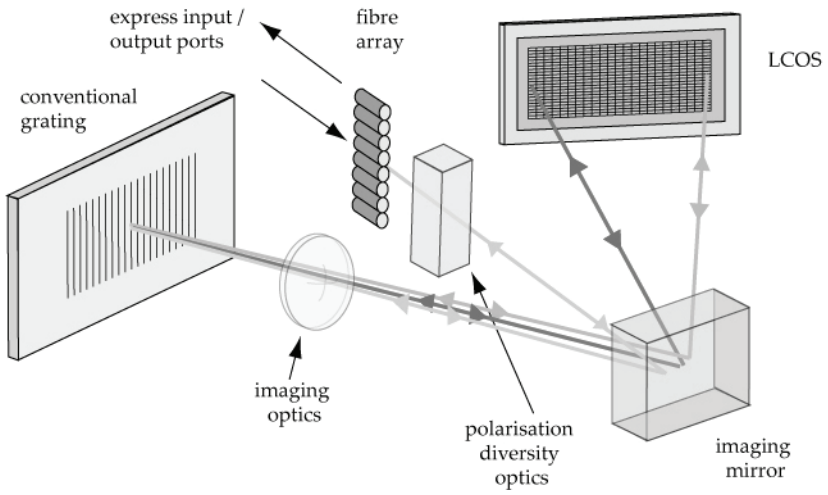


Figure 3.2 Schematic of optical design of LCoS-based Programmable Optical Processor

LCoS can be employed to control the phase of light at each pixel to produce beam-steering. In the WaveShaper, a large number of phase steps are used to create a highly efficient, low-insertion loss switch shown schematically in Figure 3.2. This simple optical design incorporates polarisation diversity, control of mode size and a 4-f wavelength optical imaging in the dispersive axis of the LCoS providing integrated switching and optical power control.

3.2 Optical Design

Light enters the device from a fibre array, and is then processed by polarisation diversity optics to align orthogonal polarisation states to maximize efficiency at the diffraction grating. The grating is designed to be at Littrow incidence, and angularly disperses the light to the LCoS array, where the reflected light is traced back through the system to the chosen output fibre, based on the beam-steering image programmed on the LCoS array.

As the wavelengths are separated on the LCoS the control of each wavelength is independent of all others and can be switched or filtered without interfering with other wavelengths.

! **Note that the WaveShaper 4000 models can only direct light of any given frequency (or wavelength) to one port at a time. Power-sharing between two or more ports is not possible.**

3.3 Operating Conditions

Environmental and Optical

The equipment shall function under the conditions outlined in Section 9: WaveShaper Specifications. The WaveShaper is designed to operate in a controlled temperature environment and if operated outside of the specified conditions, the unit may fail to operate correctly and the warranty on the unit will be void.

Electrical

The WaveShaper S-Series units are supplied with an earth mains power cable suitable for connection to the mains supply for the county of destination.

The WaveShaper M-Series module is designed to operate with OEM supplier equipment delivering 5 V-5.6 V, at the module connector. Correct operation cannot be guaranteed if the supply voltage is outside of this range. The end user is responsible for ensuring correct voltage levels. Applying voltage levels higher than 6 V, or reversing the voltage, will break the unit, and is not covered by warranty. Section 2: Getting Started documents how the M-series WaveShapers must be wired up.

Optical Safety

Care should be taken to avoid eye exposure to optical connectors when optical power is introduced to any connector whether the unit is on or off. The WaveShaper should only be operated with fibres connectors suitably terminated.

3.4 Calibration and Maintenance

The WaveShaper contains no user-serviceable components. In the event of any fault occurring, the user should contact Finisar or its representative for advice. Contact details for Finisar can be found at the front of this manual.

Calibration accuracy is tested at manufacture. There is no facility for external calibration. In the unlikely event that re-calibration is required, the unit should be returned to the manufacturer.

3.5 Storage

The WaveShaper should be stored in appropriate dry, temperature controlled conditions to prevent damage to the equipment. Exceeding the ranges defined in Table 3.1 may result in permanent damage to the equipment.

Storage Condition	Units	Min	Max	Notes
Temperature	ºC	-20	85	
Humidity	%	5	95	Non-Condensing

Table 3.1 Storage conditions

3.6 WaveShaper Optical Specification

Optical Specifications for each WaveShaper model can be found in Section 9: WaveShaper Specifications.

Section 4: WaveManager Software Overview

The WaveManager Application Suite provides a powerful graphical user interface for controlling all aspects of the WaveShaper performance. For systems requiring integration of the WaveShaper into automated test equipment, please see Section 8: WaveShaper API Programming Guide.

4.1 User Interface Overview

4.2 Explorer Panel

4.3 Adding a WaveShaper during program operation

4.4 Removing a WaveShaper during program operation

4.6 Options

4.1 User Interface Overview

The main functional sections of the WaveManager Application Suite are described below with reference to Figure 4.1

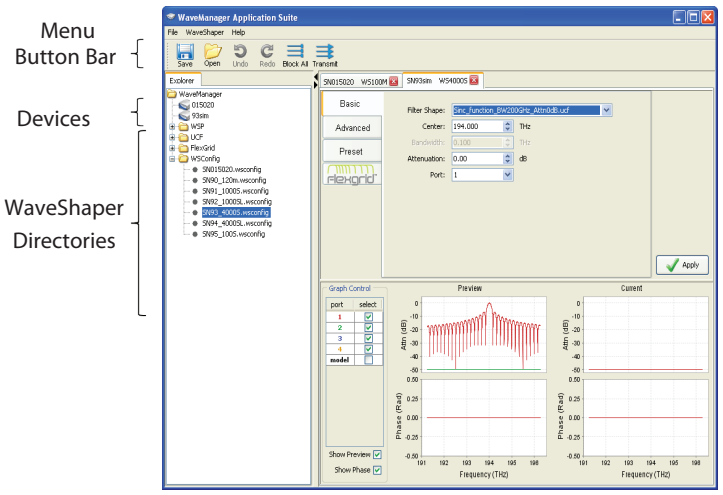


Figure 4.1 WaveManager Application Suite showing main controls

The main elements of the WaveManager Application window are:

- Menu bar - Note: file actions are based on the active control panel tab.
- Button Bar - buttons for commonly used actions based on the active control panel tab.
- Explorer control panel for easy viewing of attached and simulated devices and default directory content. Also has the ability to start a device simulation and open a *wsp* or *ucf* file for editing.

- WaveShaper Control Panel - The WaveShaper Control panel starts with automatically generated tabbed interfaces for each WaveShaper active and connected when the WaveManager application is started.

Tabbed Interface

The WaveManager Application Suite can independently control connected WaveShapers using a tabbed interface for instrument selection and control. The tab contains the WaveShaper Serial Number (eg SN001267) at its left hand side and the WaveShaper Type (eg 4000S/L) at the right hand end next to the close icon. Clicking on a tab selects the appropriate WaveShaper.

If multiple tabs are present, the focus cycles between them by pressing [CTRL+TAB].

Tabs can be closed by clicking on the close icon in the Tab area. They can be re-opened at any time by left double clicking on the device icon in the explorer panel.

Within each tab there will be one or more sub-tabs (shown down the left hand side of the tab) which provide the different ways of loading profiles to the WaveShaper. The sub-tabs available will depend on the type of WaveShaper being controlled. The operation of the sub-tabs is described in Sections 5.2 to 5.5.

Button Bar

Provides short-cuts for frequently-used commands. The actions of the Buttons are as follows:

	Save	Opens a dialog box to allow user to save the current instrument settings as a WaveShaper Preset (*.*wsp*) file.
	Open	Allows the user to open and load a saved WaveShaper Preset (*.*wsp*) file.
	Undo	Allows the user to undo the last WaveShaper profile update for the currently selected WaveShaper.
	Redo	Allows the user to redo the last WaveShaper profile update (that was previously undone) for the currently selected WaveShaper.
	Block All	"Block All" sets all frequencies to be blocked on the common port. When the "Block All" button is pressed, the necessary commands are immediately sent to the WaveShaper and the user does not need to press <Apply> to force an upload.
	Transmit	"Transmit" operates on the currently selected port of the WaveShaper and sets all frequencies to be transmitted with no attenuation. When the "Transmit" button is pressed, the necessary commands are immediately sent to the WaveShaper and the user does not need to press <Apply> to force an upload.

Graphing Area

The graphing area allows the user to view graphs which display the nominal WaveShaper profile, in terms of relative power and phase, of the selected WaveShaper. Between one and four graphs can be displayed, as set by the show preview and show phase check boxes at the left hand side of the graphing area. For a WaveShaper 4000, multiple traces can be displayed on each graph to show the status of the different ports as selected by the graph control "port" select check boxes.

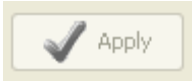
The “Current” graphs display the profile currently applied to the selected WaveShaper. The “Preview” graphs show the profile which will be generated on the selected WaveShaper when the <Apply> button is pressed.

The graph control "model" check box provides for theoretical modelling of the current filter shape as shown in the “Current” graph.

The user can zoom in on an area of graph by holding down either mouse button and dragging the mouse from left to right to define a zoom area. The default graph axes are restored by holding down either mouse button and dragging the mouse from right to left.

Apply Button

The <Apply> button applies to the currently selected WaveShaper any changes made to the WaveShaper profile, with the progress bar indicating progress in uploading the new profile. The button will change appearance to show connectivity and update status as shown in the table below.

	click to apply changed settings to the device.
	current settings have been applied to the device.
	communication to the device has been (temporarily) lost. Click apply to try and re-establish communication. If this fails, the reason for failure will be displayed in an error message.

Menu Structure

The WaveManager Application Suite provides access to many functions through drop-down menus as defined below.

File

Save	Opens a dialog box to allow the user to save the current instrument settings as a WaveShaper Preset (*.wsp) file (see Section 6.3). Not available for 100s or 120s.
Save a Copy ...	Opens a dialog box to allow the user to save the current open file (in the built-in text editor) to a new file name. Only available in file-edit mode.
Open	Allows the user to open and load a saved WaveShaper Preset (*.wsp) file (see Section 6.3). Not available for 100s.

Import

Import User Configured Filter (UCF) : Imports a **.ucf* file into the current ucf directory.

Import WaveShaper Preset (WSP): Imports a **.wsp* file into the current wsp directory.

Save Model — Opens a dialog box to allow user to save the current theoretical model as a WaveShaper Preset (**.wsp*) file (see Section 6.3). Not available for 100s or 120s.

Open WaveManager Data Folder

Opens windows explorer at the Users WaveManager Application data folder (C:\Documents and settings\User name\Application Data\WaveManager\).

Exit

Exits the program in the same manner as clicking the "x" on the menu bar.

WaveShaper

Options — Allows the user to set the various options and file locations for the Application Suite. See Section 4.6 for details.

Add WaveShaper Configuration.
Adds a **.wsconfig* file to the list of supported WaveShapers.

In nm — Allows the user to select display/enter values in nm or THz.

Run WaveSketch — Opens the WaveSketch Application in a new window.

Run Fourier Processor — Opens the WaveShaper Fourier Processor Application for power splitting in a new window.

Help

WaveManager Help — Opens the WaveManager Application Suite User Manual.

About — Displays information about the WaveManager Application Suite release version.

4.2 Explorer Panel

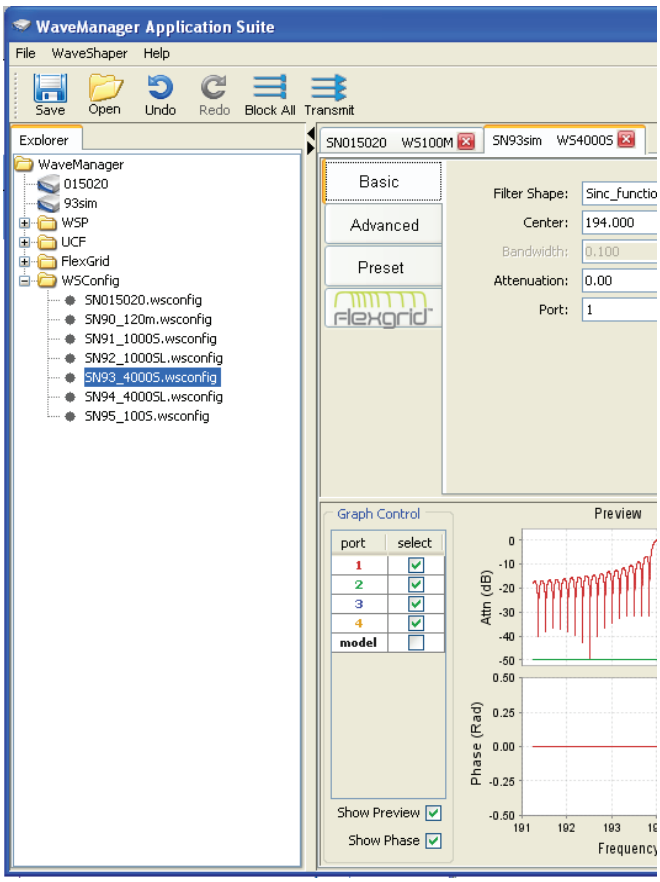


Figure 4.2 The Explorer Panel

The explorer panel is populated at start up with icons representing connected Waveshaper devices and the Waveshaper default directory tree. The Show/Hide buttons allow the user to collapse this panel leaving more space for the WaveShaper Control Panel. Select and drag the Panel Width Control bar to adjust the width of the panel.

To run a simulation of a device, select the *wsconfig* file of the device for simulation, right click on the mouse, and select the run simulation pop up. The active icon for this serial number will have the letters sim added to highlight that it is a simulation running.

Any device tab may be closed by clicking on the close symbol in the device's control tab. Any device that has been connected or simulated is available in the Explorer Panel until the WaveManager application is exited. A device's tab can be re-opened for display at any time by double left clicking on the device icon in the explorer panel.

Left double clicking on any WaveShaper Preset (**.wsp*) file or User Configured Filter (**.ucf*) file in the explorer window opens a tabbed text editor for the file in the WaveShaper Control Panel. Changes to the file can be saved by the Save button or through the File/Save menu.

4.3 Adding a WaveShaper during program operation

The WaveManager Application Suite continuously monitors for the presence of additional WaveShapers and will automatically add new units to the program up to the limit of machines supported.

If a WaveShaper is attached that has not been previously installed and has firmware supporting embedded configuration files, the configuration file will be automatically downloaded (Figure 4.3)and stored in the appropriate directory.

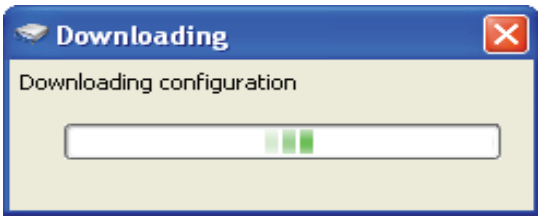


Figure 4.3 WaveShaper Embedded Configuration File Download Indicator

If the download is unsuccesful or not available , there will be a request for the configuration file for the new WaveShaper (Figure 4.4). This file is provided on the CD or USB Drive supplied with the WaveShaper.

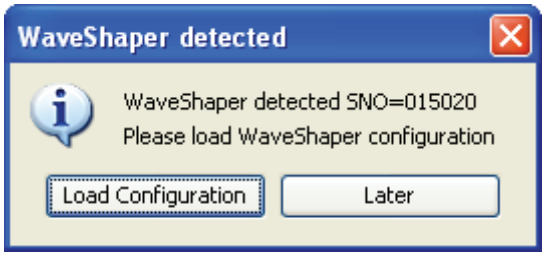


Figure 4.4 Dialog box requesting WaveShaper Configuration file for a new WaveShaper

Clicking on <Load Configuration> will open up the dialog box in Figure 4.5. Select the appropriate WaveShaper configuration file (which can be identified by the suffix **.wsconfig*) and then press <Open> to load the configuration file. If you do not wish to load the configuration file at this time, then press <Later>. WaveShaper configuration files can be added at any time using the WaveShaper/Add WaveShaper Configuration menu option.

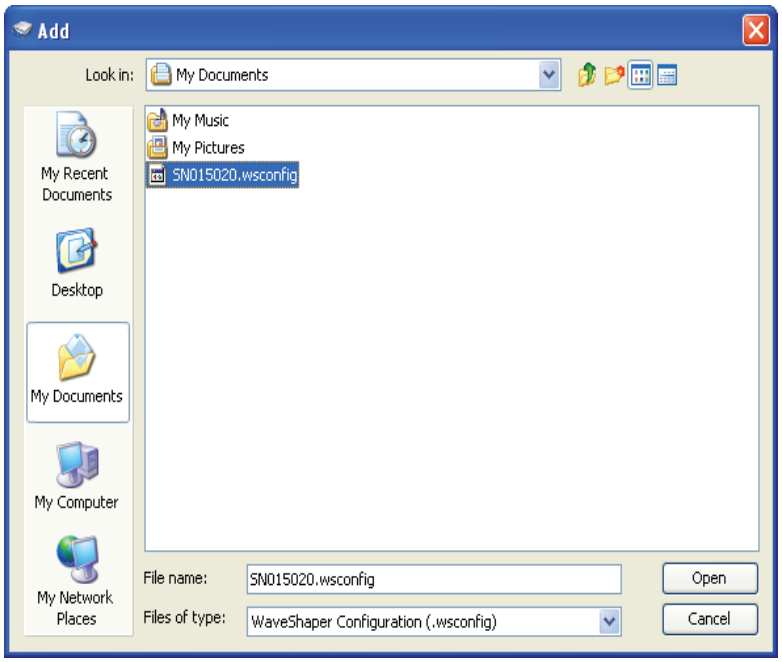


Figure 4.5 Add WaveShaper Configuration File Dialog box.

4.4 Removing a WaveShaper during program operation

As the WaveManager Application Suite will continuously monitor the presence of all WaveShapers, it can also detect the removal (either through power-down or disconnection) of a WaveShaper. The "Apply" button on the disconnected WaveShaper 's tabbed interface changes to 

. As with Simulations, the Tabbed interface can be closed and re-opened later by double clicking on the device icon in the explorer panel.

4.5 Multiple configuration file for 16000 units:

The WaveShaper 16000 supports multiple different port configurations (1x19, 4x16, 8x12 and 10x10) which are enabled by loading different configuration files into WaveManager. The default configuration (embedded on the 16000 chassis) is a 4x16 configuration.

Other configuration files are stored on the CD or USB drive provided with the WaveShaper and use a consistent naming format, for example SN086180_10x10.wsconfig for a 10x10 configuration. This naming format must be maintained to ensure correct operation of the WaveShaper 16000.

The additional configuration files must be copied to the wsconfig directory by using the File -> Open WaveManager Data Folder command.

When there are multiple configurations for the same unit in the wsconfig directory, the user will be prompted to select the required wsconfig file when the WaveShaper 16000 is connected as shown in Figure 4.6.

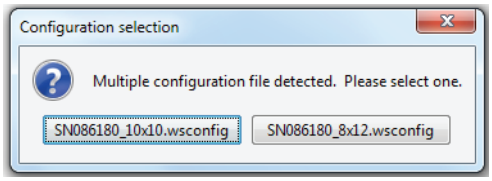


Figure 4.6 Select WaveShaper Configuration File Dialog box for WaveShaper 16000

4.6 Options

The user-settable Options (accessible through the WaveShaper/Options menu) allow the user to define the locations of the various WaveShaper files. For Windows systems, these are as outlined in Table 4.1. For details on file formats etc, see Sections 6.2 to 6.4.

File Type	Default Location
User Configured Filter (*.ucf)	C:\Documents and Settings\User Name\ Application Data\WaveManager\ucf
WaveShaper Preset (*.wsp)	C:\Documents and Settings\User Name\ Application Data\WaveManager\wsp
Flexgrid (*.wsgrid)	C:\Documents and Settings\User Name\ Application Data\WaveManager\wsgrid
WaveShaper Configuration (*.wsconfig)	C:\Documents and Settings\User Name\ Application Data\WaveManager\wsconfig

Table 4.1 Summary of Options settings.

5.1 Overview

5.2 Basic sub-tab

5.3 Advanced sub-tab

5.4 Preset sub-tab

5.5 Flexgrid sub-tab

Section 5: Controlling the WaveShaper Spectrum with the WaveManager Application Suite

5.1 Overview

The WaveShaper profile is generated by the WaveManager Application Suite and uploaded to the currently selected WaveShaper when the <Apply> button is pressed. This section describes the different ways in which the profile on the WaveShaper can be modified.

For ease of use, the different ways of modifying the optical spectrum are addressed using different sub-tabs, located at the left hand side of the main WaveShaper tab. The sub-tabs available will depend on the type of WaveShaper being controlled and the available sub-tabs are summarized in Table 5.1.

Name	Description	Supported by	Section
Basic	Controls basic WaveShaper functionality	All Models, except 120m and 120s.	5.2
Advanced	Allows the user to apply additional signal conditioning with phase, delay or dispersion offsets over the full spectrum or a given window with the ability to select the number of repeats	All models except 100s, 100m, 120s and 120m.	5.3
Preset	Allows the user to load Preset WaveShaper Configurations	All models except 100m and 100s.	5.4
Flexgrid	Emulation of *Flexgrid* functionality	4000 and 16000 models only	5.5

Table 5.1 Summary of WaveShaper sub-tabs

In all cases, the changes which are made on the sub-tab are reflected in real-time on the "Preview" pane (if visible) but are only uploaded to the WaveShaper when the <Apply> button is pressed.

5.2 Basic sub-tab

The Basic sub-tab provides an easy-to-use interface to control the profile of a WaveShaper using basic filter shapes (Figure 5.1).

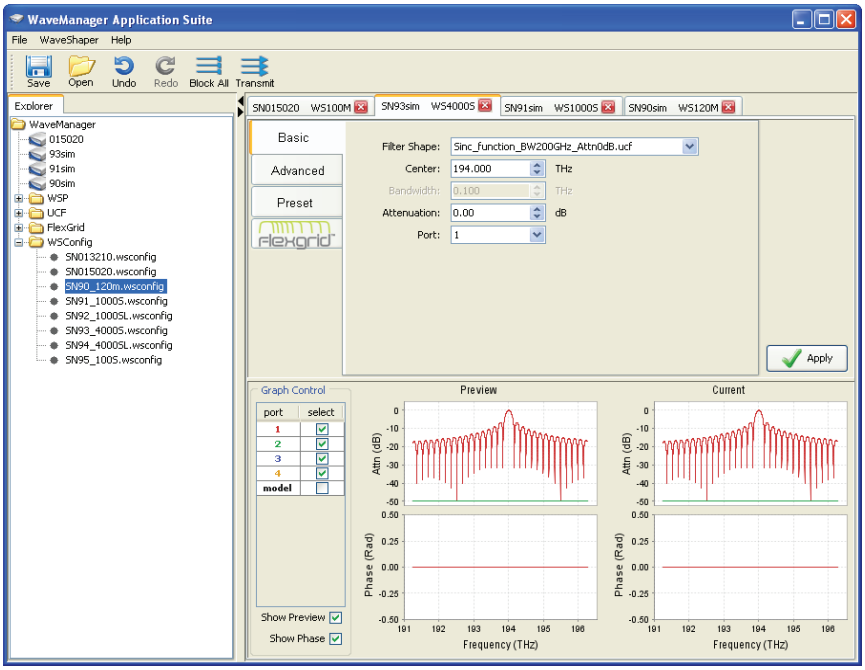


Figure 5.1 Screenshot of Basic sub-tab

5.2.1 Filter Shape

This allows the user to select the filter shape from a drop-down list.

For all WaveShapers except the 100S, all pre-programmed filters (Bandpass, Gaussian, Bandstop) and installed User Configured Filters (UCFs) are available. The UCFs displayed are those stored in the user-defined UCF directory (set in the Options). For more information on UCFs, see Section 6.2.

For the 100S, only Bandpass and Gaussian filter shapes are available.

5.2.2 Centre Frequency Control

This allows the user to set the centre frequency of the filter and is programmable in 1 GHz steps.

Adjustment of the Centre Frequency is through direct editing in the entry box or through the use of the up/down arrows associated with the entry box. The value which can be entered is limited to the operating range of the unit.

5.2.3 Bandwidth Control

This allows the user to change the bandwidth of the filter selected in 1 GHz increments.

The display shows the nominal 3 dB bandwidth selected. The change in the bandwidth is symmetric to within +/- 2 GHz around the centre of the filter.

Adjustment of the Bandwidth is through direct editing in the entry box or through the use of the up/down arrows associated with the entry box. See section 9.3 Optical Specifications for the bandwidth control range for specific models .

Note: The bandwidth control is accurate for values down to approximately 20 GHz, below this value the bandwidth approaches the instrument's optical transfer function of approximately 10-12 GHz.

5.2.4 Attenuation Control

The Attenuation Control provides a power control function for the current filter.

Adjustment of the Attenuation is through direct editing in the entry box or through the use of the up/down arrows associated with the entry box. The value which can be entered is limited to 35.0 dB to match the operating range of the unit.

Attenuation control is not available for the 100S.

5.2.5 Port Selector

The Port Selector Control is only available for the 4000 and 16000 range of WaveShapers.

For the 4000, this allows the user to select the port to which the output is directed. The default value after start-up is Port 1.

For the 16000, in 4 x 16 configuration, the user can to select the input port (A - D) and the port to which the output is directed (1-16). The default values after start-up are Input Port A and Output Port 1.

Additional 16000 NxM configurations uses two parts port naming as shown in below:

4x16 : [A…D] – [1…16]

10x10: [A…D1…6] – [7…16]

8x12: [A…D1…4] – [5-16]

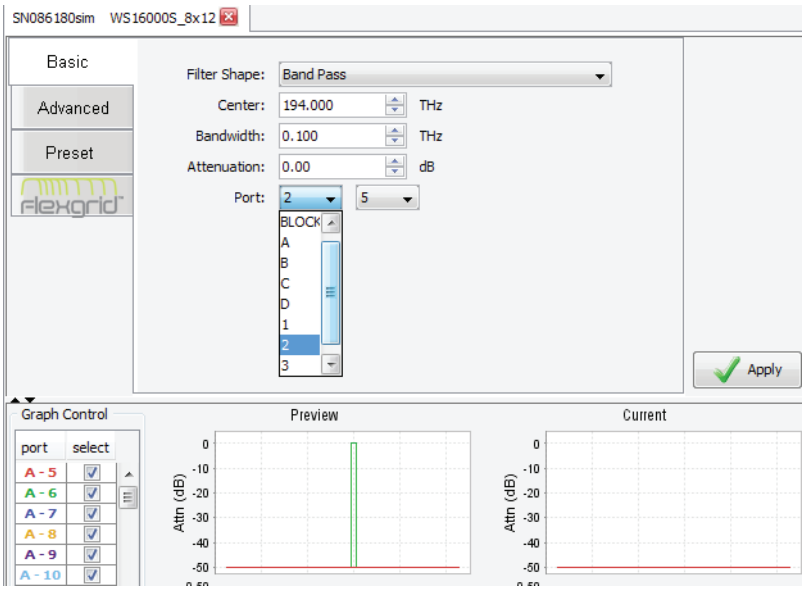


Figure 5.2 Port Selection for the 16000 in 8 x 12 Configuratino

5.3 Advanced sub-tab

The Advanced sub-tab allows the user to also control phase, delay or dispersion. This control can be set across the full spectrum or within a specified window centred at the filter centre and with a specified repeat frequency and number of occurrences.

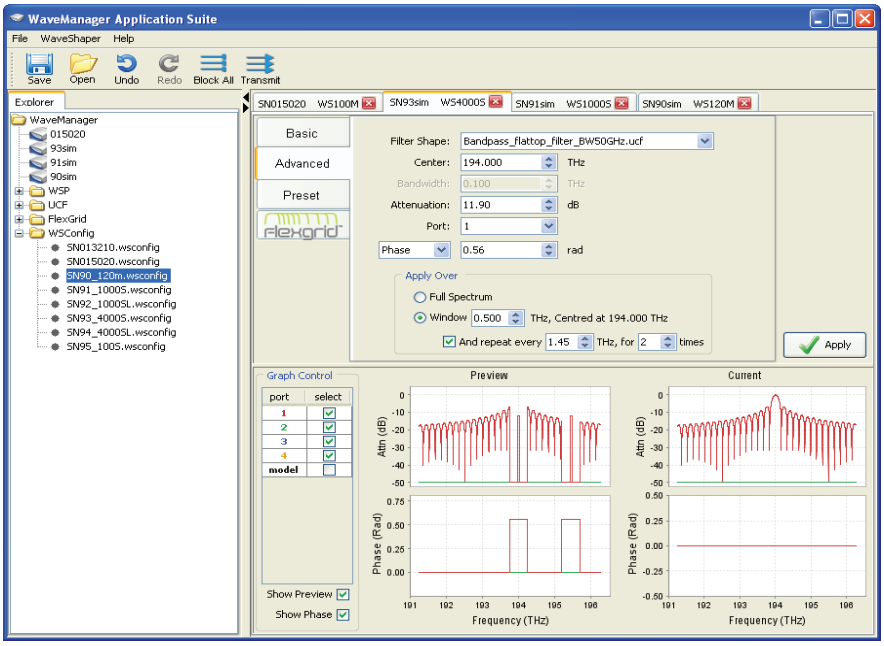


Figure 5.3 Screenshot of Advanced sub-tab.

5.3.1 Phase/Delay/Dispersion Control

This drop down list box lets the user select which measurement value they wish to enter.

Select Control	Measured in	Control Range
Phase	rad	-2π to 2π
Delay	ps	-25 to 25
Dispersion	ps/nm	-100 to 100

Adjustment of the Phase/Delay/Dispersion Value is through direct editing in the entry box or through the use of the up/down arrows associated with the entry box.

5.3.2 Apply Over

This control allows the user to control the spectral range over which a filter function is applied.

The Window function is used to limit the applied range of a UCF and also to define the region of the frequency spectrum which is updated when the <Apply> button is pressed.

The control allows the user to set the 6 dB window applied to the selected filter in 1 GHz increments from 10 GHz up to the operating range of the unit. The change in the window is symmetric around the centre of the filter. Adjustment of the Window is through direct editing in the entry box.

Note: The Window control is accurate for values down to approximately 20 GHz, below this value the bandwidth approaches the instrument transfer function of approximately 10-12 GHz .

The repeat function allows the user to specify repeat applications of this filter at a specified frequency gap for a specified number of times (up to the operational limit of the unit).

The Window function is defined at 6 dB as this is the value which is required to ensure that the function can be stacked one-against-the-other for use when switching signals using the WaveShaper 4000 or 16000 to emulate a WSS. For a mathematical derivation of this, see the White paper entitled "*Filter Bandwidth Definition of the WaveShaper S-series Programmable Processor*" available on the Finisar website. This functionality also provides backwards-compatibility with the earlier releases of WaveShaper software which used the windowing function (called as the filter bandwidth) at all points in the setting up of a filter.

The "Apply Over" control is not available for the 100S.

5.4 Preset sub-tab

The Preset sub-tab allows the user to load pre-defined WaveShaper profiles which define the complete spectral profile of the instrument in terms of relative power, phase and port selection across the entire operating bandwidth of the instrument.

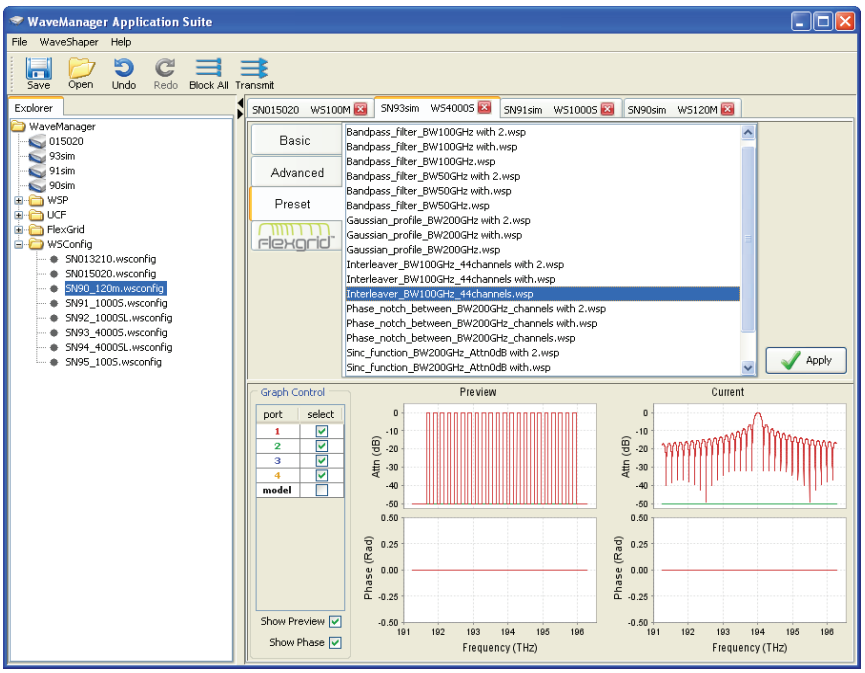


Figure 5.4 Preset sub-tab

Figure 5.4 shows a screenshot of the Preset sub-tab. This drop-down file list allows the user to select a **.wsp* file which is then loaded into the WaveShaper when the <Apply> button is pressed. The file format used is the **.wsp* (WaveShaper Preset) which is explained in more detail in Section 6.3.

The "Preset" sub-tab is not available for the 100S.

5.5 *Flexgrid* sub-tab

5.5.1 Introduction to Flexgrid

The *Flexgrid* sub-tab allows the user to set up a *Flexgrid* channel-based grid to emulate certain aspects of Finisar's Wavelength Selective Switches (WSS) using 400 x 12.5 GHz channel slices which can be combined together to generate the required Channels.

Each defined Channel must be between 2 and 16 slices wide (25 to 200 GHz) and can be controlled by switching to a port, attenuating or blocking it. Each Channel must be assigned a unique Channel ID between 1 and 256. All slices must be either part of a Channel or Unassigned.

All unassigned slices are set to block and are grouped into contiguous spectral regions, each of which is given Channel ID = 0. There is no upper or lower limit on the size of an Unassigned spectral region.

A channel can be modified by *reducing* its width (in which case the 'discarded' parts of the channel are redefined as "Unassigned" or by *increasing* its width by extending into adjacent Unassigned spectral regions. A channel cannot be expanded by overwriting adjacent channels. These must first be removed (set to Unassigned) and some or all of the Unassigned spectrum can then be added to the existing channel.

In *Flexgrid* mode, the WaveShaper attenuation range is limited to a maximum of 20 dB, rather than the 35 dB available in the standard WaveShaper mode. The *Flexgrid* sub-tab is only supported for WaveShaper 4000S and 4000M (C-band).

5.5.2 Overview of the Flexgrid sub-tab

A *Flexgrid* channel plan can be generated using either the panel controls or from a *Flexgrid *.wsgrid* tab-delimited input file (Section 6.4).

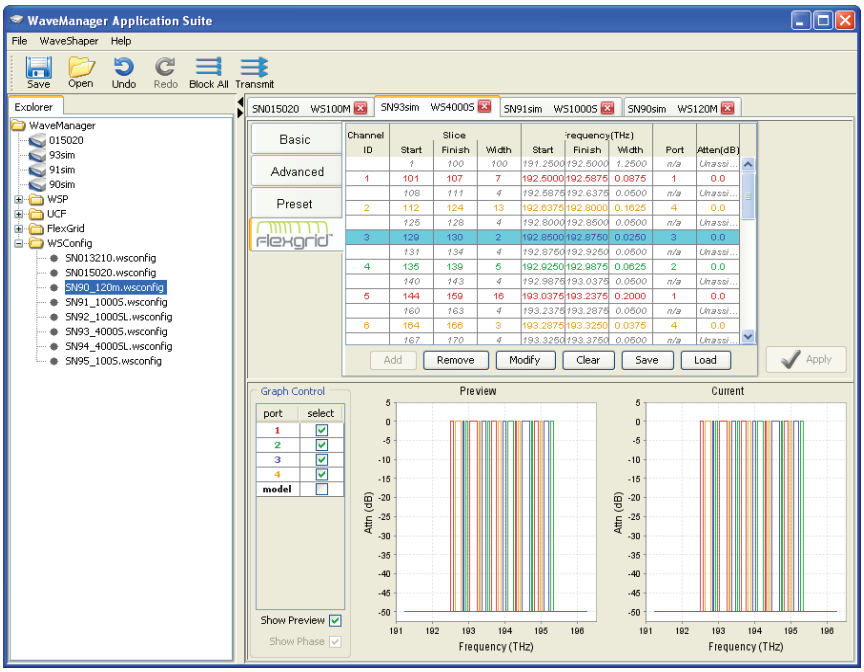


Figure 5.5 Flexgrid sub-tab showing display table with multiple Flexgrid channels defined

The display table (Figure 5.5) shows the current grid settings and changes to the channel allocation are made by selecting the appropriate row(s) and either using the Add/Remove/Modify *Flexgrid* controls as defined below or double-clicking on the selected row.

<Add>

This button allows the user to create a new channel from within an unassigned region of the spectrum.

<Remove>

This button allows removes a channel and de-assigns its spectrum.

<Modify>

Modifies the channel parameters using the Channel Definition Dialog Box. Note that if the selected rows in the display table include any unassigned spectral region(s) then these are ignored by the dialog box, but it will continue to operate correctly on the defined channels. This allows, for instance, the user to select the whole table (by clicking on the Channel ID header) and then set all defined channels to the same attenuation and port.

<Save>

This button allows the user to Save the current Grid Display Table as a **.wsgrid* file.

The button is greyed out if there is no Grid Display Table to save or if the table has not been changed since the last save.

The format of the **.wsgrid* file is defined in Section 6.4.

<Load>

This button allows the user to load a **.wsgrid* file in the Grid Display Table.

The format of the **.wsgrid* file is defined in Section 6.4.

<Clear>

This button clears the currently defined *Flexgrid* allocation and returns all slices to the unassigned state. To prevent loss of data, the system will warn the user before allowing the defined *Flexgrid* channel plan to be deleted.

5.5.3 Setting the Parameters for a Single Flexgrid-Channel

The parameters for a *Flexgrid* channel can be set by one of four methods:

1. Selecting an existing channel in the Grid Display Table and clicking the <Modify> button.

2. Selecting an unassigned region of the spectrum in the Grid Display Table and clicking the <Add> button.

3. Double clicking on an existing channel

4. Double clicking on an unassigned region of the spectrum

In all cases, a dialog box (Figure 5.6) is used to set or modify the parameters of a selected channel.

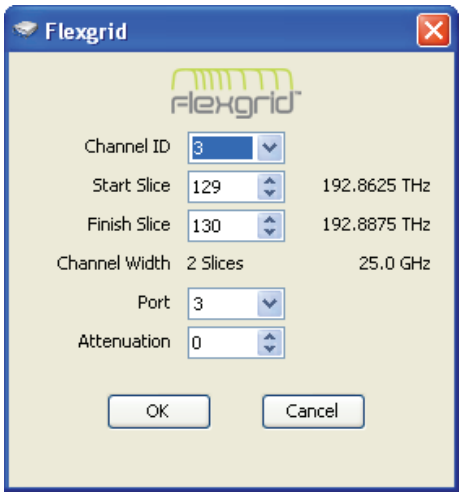


Figure 5.6 Flexgrid Channel Definition Dialog Box

The controls in the Channel Definition Dialog Box operate as follows:

Channel Number

Every defined channel must be assigned a unique channel number in the range 1 to 256. This drop-down list shows all unused and available channel numbers. When a channel is deleted or unassigned, the channel number becomes available for re-use. Channel numbers do not have to be contiguous.

Start Slice and Stop Slice

These drop down boxes allow the user to specify the start and finish slices for a Flexgrid Channel. The equivalent Centre Frequencies for each slice are shown to the right of the control whilst the channel width is indicates below the Finish Slice control

Port

For a WaveShaper 4000, this allows the user to set the output port for the Selected Channel.

For a WaveShaper 16000, this allows the user to select both the input and output ports for the Selected Channel.

A "Block" port is also available.

Attenuation

Allows the user to set the Flexgrid Channel Attenuation in the range 0 - 20 dB.

5.5.4 Setting the Parameters for Multiple Flexgrid-Channels

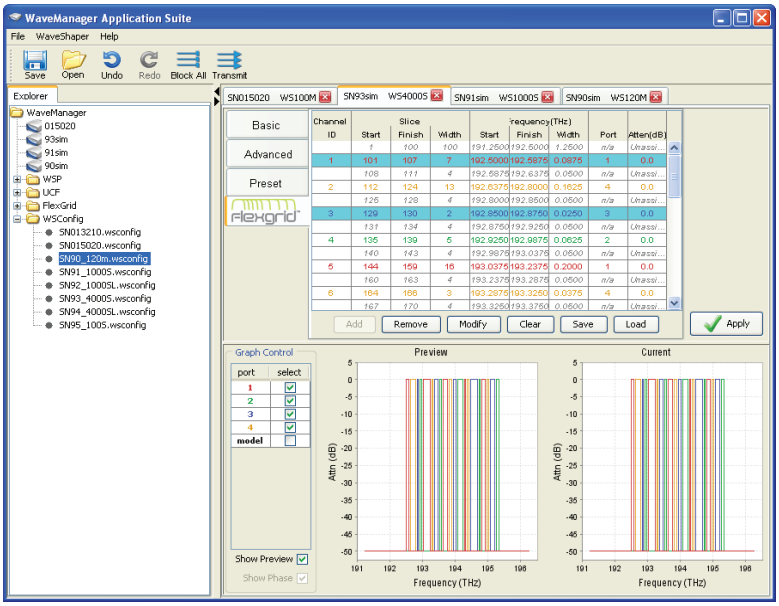


Figure 5.7 Flexgrid sub-tab showing multiple rows selected

In some cases it is convenient to be able to set certain parameters for multiple channels at the same time, when switching between ports or changing attenuation, for example. The *Flexgrid* sub-tab (Figure 5.7) allows the user to select multiple rows of a *Flexgrid* Channel Plan by holding down the CTRL key and selecting the required rows of the Grid Display Table and then clicking on the <Modify> button to vary the Port and Attenuation controls for all the channel simultaneously (Figure 5.8).

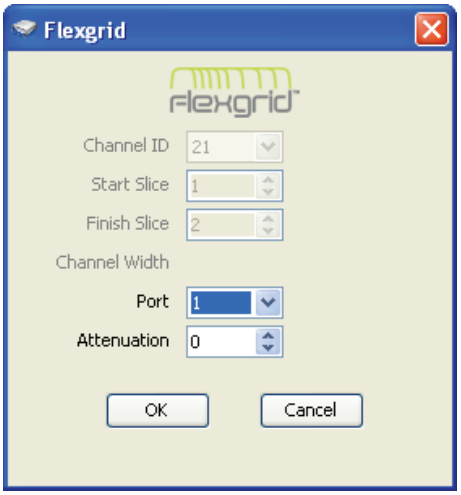


Figure 5.8 Flexgrid Channel Dialog Box when multiple channels selected

Section 6: WaveShaper File Formats

6.1 Overview

6.2 User Configured Filters (.ucf) files*

6.3 WaveShaper Preset Files (.wsp) files*

6.4 Flexgrid (.wsgrid) files*

6.5 WaveShaper Preset Power Split Profile (.psp) files*

6.1 Overview

The WaveManager software supports three different input file formats; **.ucf*, **.wsp* and **.wsgrid*. These have different applications as outlined in Table 6.1.

File Type	File Suffix	Description	Supported by	Section
User Configured Filter	**.ucf*	Controls basic WaveShaper Functionality	All models except 100S	6.2
Preset	**.wsp*	Allows the user to load Preset WaveShaper Configurations		6.3
Flexgrid™	**.wsgrid*	Emulation of Flexgrid functionality	4000 and 16000 models only	6.4

Table 6.1 Summary of WaveShaper File Types

6.2 User Configured Filters (**.ucf*) files

The ability to load and then easily control user configured filters is key to the operation of the WaveShaper. To this end, the WaveManager supports the same *.ucf* file import format used in previous versions of the WaveShaper software.

To ensure data integrity, the software will parse and truncate/interpolate the *.ucf* file to ensure the requested filter shape is calculated to conform to the limits set by the WaveShaper capabilities. The following rules for preparing and interpreting the *.ucf* files therefore apply.

The number of frequency data points must be at least one. The interpolation in the WaveManager software chooses the attenuation and phase values corresponding to the frequency value that is nearest to the respective frequency in the WaveShaper. Hence, if there is only one point defined, all frequencies will be set to the same attenuation and phase values defined by that point. Any WaveShaper frequency outside of the defined frequency range of the input file will be set to the respective edge points of the input file definition, as that will be the nearest neighbour to these frequencies.

For optimal consistency and portability of *.ucf* files between different WaveShapers, it is recommended to specify the profiles down to a resolution of 1 GHz.

6.2.1 File Structure

The format of a *.ucf* file is a tab delimited text file with three columns: Frequency Offset (THz), Attenuation (dB) and Phase (Rad). The file can be generated using a spreadsheet program, such as Microsoft Excel (saving the document as a **tab delimited text file**). The file extension must be "*.ucf*" for the file to appear in the File Open dialog box.

6.2.2 Frequency

The filter frequency is defined in THz as the detuning (positive and negative) from an arbitrary centre frequency. The frequencies do not have to be balanced around the centre frequency, but the frequency data set must contain one, and only one, value of zero, which the WaveShaper interprets as the centre frequency. The values in the frequency column must also be monotonically increasing. Each frequency data point must have a corresponding value of Attenuation and Phase. If no specific attenuation or phase value is required, values of 0 must be specified at these points.

6.2.3 Attenuation

The calibrated attenuation values available in the WaveShaper hardware are 0 to 30 dB. (0-10dB for the 120 models) However, the WaveShaper hardware will control attenuation down to (maximally) 40 dB and also provides a 'Block' state with attenuation of typically >50 dB. To guarantee the portability of filter shapes between WaveShaper units, the **.ucf* attenuation range should be limited to 0-30 dB. The interpretation of the **.ucf* files provides access to the full range of attenuation controls as follows:

Requested Attenuation	**.ucf* interpretation
<0 dB	Truncated to 0 dB
0 - 30 dB (Note 0-10dB for 120)	Guaranteed accuracy according to the WaveShaper specification
30.1 - 40 dB	WaveShaper attempts to set attenuation to requested value. No guarantee of accuracy (generates error response for 120 models)
>40 dB	Signal set to 'Block' (generates error response for 120 models)

Table 6.2 Summary of attenuation interpretation for user configured filter files

6.2.4 Phase

The phase control range available in the WaveShaper is 0-2π. (Not available for 120 models.) The User Configured Filter may specify a phase outside of this range, however, this will be re-calculated by the WaveShaper software as input phase mod *2π* shown in Figure 6.1 below.

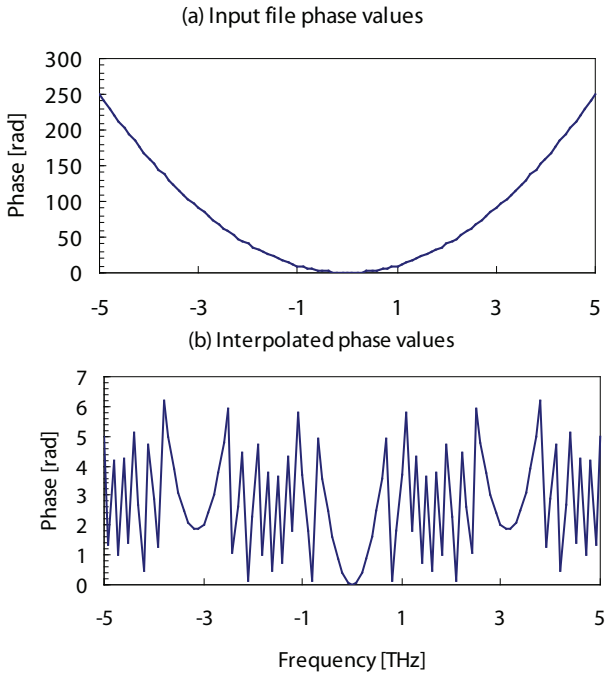


Figure 6.1 Example of limits imposed by interpolation of phase values in the .ucf file. The phase values are calculated as (input phase mod 2π).

The mathematical operations performed by the WaveShaper software on the User Configured Filter data file are summarised in Table 6.3 below.

Parameter	Units	Interpretation
Frequency	THz	Interpolated to fit defined filter (channel) bandwidth
Attenuation	dB	Value dependent - see Table 6.2
Phase	Rad	Modulo 2π

Table 6.3 Summary of data interpretation for user configured filters

Examples of **.ucf* files are provided with the software and should be studied to understand the structure of the file.

Note: There is a degree of inter-linking between the amplitude and phase contouring and varying one will have some effect on the other. In particular, applying large amounts of dispersion to a channel can have a significant impact on the channel amplitude shape. For full details, see M.A.F. Roelens, J. Bolger, G. Baxter, S. Frisken, S. Poole and B.J. Eggleton, "Dispersion Trimming in a Reconfigurable Wavelength Selective Switch", in *Journal of Lightwave Technology*, vol. 26, p.73, 2008.

6.2.5 Using Microsoft Excel to generate **.ucf files*

When using Microsoft Excel to generate a **.ucf* file, the following points should be noted.

1. The file should be saved as Text (Tab delimited) (**.txt*) file and subsequently renamed to have the suffix "*.ucf*".

2. Before saving as a **.txt* file, the format of all columns should be set to 'Number'

with at least three (3) decimal places. If the number format is set to less decimal places, Excel may misinterpret the data during the translation to a tab delimited file.

6.3 WaveShaper Preset Files (*.*wsp*) files

Preset files provide a way to fully define the filter/switch profile of a WaveShaper across all Frequencies and Ports. As such it is a very powerful tool to easily configure the WaveShaper to a required complex spectrum.

To ensure data integrity, the WaveManager software parses the *.wsp* files to ensure the requested filter shape is calculated to conform to the limits set by the WaveShaper capabilities. The following rules for preparing and interpreting *.wsp* files therefore apply.

6.3.1 File Structure

The format of a *.wsp* file is a tab delimited text file with four columns: **Absolute** Frequency (THz), Attenuation (dB), Phase (Rad) and Port. The file can be generated using a spreadsheet program, such as Microsoft Excel (saving the document as a tab delimited text file). The file extension must be "*.wsp*" for the file to appear in the File Open dialog box.

6.3.2 Number of Frequency Data Points

Unlike *.ucf* files, every GHz must be specified for the whole available spectrum in every *.wsp* file. The first and last points in the file must be must be within the frequency range as per specification table in 9.3 on page 83 for the respective WaveShaper model. A partial definition that covers a continuous range within the valid frequency range is also allowed. To avoid interpolation and rounding problems, it it recommended to check that the frequencies are specified to at least 3 decimal places in the *.wsp* file. This means for example that the first few lines in the *.wsp* file for a 4000S model might look as follows:

```
191.250          2.0     3.14       2
191.251          2.1     3.15       2
191.252          2.2     3.16       2
191.253          2.2     3.17       2
191.254          2.2     3.18       2
...
```

The last few lines for the same model could then look as follows:

```
...
196.271          4.0     3.18       4
196.272          5.1     3.17       4
196.273          6.2     3.16       4
196.274          7.2     3.15       4
196.275          8.2     3.14       4
```

6.3.3 Frequency

Each frequency data point (specified in absolute terms for *.wsp* files) must have a corresponding value of Attenuation, Phase and Port. If no specific attenuation or phase value is required, values of 0 must be specified at these points.

6.3.4 Attenuation

The calibrated attenuation values available in the WaveShaper hardware are 0 to 30 dB. (0-10 dB for 120 models). However, the WaveShaper hardware will control attenuation down to (maximally) 40 dB and also provides a 'Block' state with attenuation of typically >50 dB. To guarantee the portability of filter shapes between WaveShaper units, the attenuation range should be limited to 0-30 dB. However, the interpretation of the **.wsp* files provides access to the full range of attenuation controls as follows:

Requested Attenuation	**.wsp* interpretation
<0 dB	Truncated to 0 dB
0 - 30 dB	Guaranteed accuracy according to the WaveShaper specification. Limited to 0-10 dB forWaveShaper 120
30.1 - 40 dB	WaveShaper attempts to set attenuation to requested value. No guarantee of accuracy. (Generates error response for 120 models.)
>40 dB	Signal set to 'Block' (generates error response for 120 models)

Table 6.4 Summary of attenuation interpretation for *.wsp *files*

6.3.5 Phase

The phase control range available in the WaveShaper is 0-2π. (Not available in the 120 models, phase must be is set as 0.) The **.wsp* file may specify a phase outside of this range, however, this will be re-calculated by the WaveShaper software on interpolation as (phase modulo 2π) as shown in Figure 6.1.

6.3.6 Port

The port range available depends on the WaveShaper model, as outlined in Table 6.5 below

120	Port must be set to 1 and the block state is achieved by using the ws_load_predefined profile function described in Section 8.4.7.
1000	Port needs to be either 1 or 0, where 0 corresponds to the block state, and 1 to the only available output port.
4000	Output port can be set to 0, 1, 2, 3, or 4 , where 0 corresponds to the block state.
16000	The alpha-numeric port combination (eg A-1) is referenced by a sequential port number as specified in Table 6.6 below. Block is again signified by port 0.

Table 6.5 Port definition for use in WSP Files

4 x 16 configuration – 64 ports			8 x 12 configuration – 96 ports		
Input	Output	Port #	Input	Output	Port #
A	1	1	A	5	1
…	…	…	…	…	…
A	16	16	A	16	12
B	1	17	B	5	13
…	…	…	…	…	…
B	16	32	B	16	24
C	1	33	C	5	25
…	…	…	…	…	…
C	16	48	C	16	36
D	1	49	…	…	…
…	…	…	…	…	…
D	16	64	4	5	85
			…	…	…
			4	16	96
10 x 10 configuration – 100 ports			1 x 19 configuration – 19 ports		
Input	Output	Port #	Input	Output	Port #
A	7	1	A	B	1
…	…	…	A	C	2
A	16	10	A	D	3
B	7	11	A	1	4
…	…	…	A	2	5
B	16	20	…	…	…
C	7	21	A	15	18
…	…	…	A	16	19
C	16	30			
…	…	…			
…	…	…			
6	7	91			
…	…	…			
6	16	100			

Table 6.6 Port Allocation for WaveShaper 16000

6.4 Flexgrid (*.*wsgrid*) files

The *.wsgrid* file allows the user to generate arbitrary grid patterns with mixed channel widths, etc. The file saved in Extensible Markup Language (XML) format. An XML Schema Definition (wsgrid.xsd) file is included in the installation directory, as well as a few sample .wsgrid files. It is recommended to use the WaveManager software interface as the editor for these files. The table in the WaveManager software contains columns as defined in Table 6.7. Note that all slices must be defined in the *.wsgrid* file or the software will refuse to load and generate an "Incompatible or Corrupted *.wsgrid* file" error.

It is recommended to generate an example of a *.wsgrid* file with the WaveManager software, and then use that file as a template that can be adapted.

Column	Description	Values	Comments
1	Channel ID	0 or 1-256	Each channel number must be unique, but channel numbers do not have to be consecutive or monotonic. Unassigned regions of the spectrum must be numbered "0" and multiple unassigned regions are allowed.
2	Channel Start Slice	1-399	First slice of channel.
3	Channel Finish Slice	2-400	Channel width must be > 25 GHz
4	Attenuation	0 – 20 or "Block"	Only 20dB attenuation supported in flexgrid™ mode
5	Port	1 – 4 or 1 - 100	WaveShaper 4000 WaveShaper 16000 (see Table 6.6

*Table 6.7 Format of *.wsgrid files*

6.5 WaveShaper Preset Power Split Profile (*.*psp*) files

Preset files provide a way to fully define the filter/power split profile of a WaveShaper across all Frequencies and Ports. It is the power splitting equivalent of the WaveShaper Preset File (*.*wsp*) and provides frequency, attenuation and phase information for all ports undergoing power splitting.

As such it is a very powerful tool to easily configure the WaveShaper to a required complex, multiport spectrum.

To ensure data integrity, the WaveShaper Fourier Processor software parses the *.*psp* files to ensure the requested filter shape is calculated to conform to the limits set by the WaveShaper capabilities. The following rules for preparing and interpreting *.*psp* files therefore apply.

6.5.1 File Structure

The PSP file format can be broken up into two sections: the header and the body.

The file can be generated using a spreadsheet program, such as Microsoft Excel (saving the document as a tab delimited text file). The file extension must be "*.psp*" for the file to appear in the File Open dialog box.

6.5.2 PSP Body

The format of a *.psp* file is a tab delimited text file with up to 9 columns: **Absolute** Frequency (THz), Attenuation (dB), Phase (Rad) ... [Attenuation (dB), Phase (Rad)].

The PSP body is expected to have a total of 9 columns unless a port allocation vector is given (see 6.5.3 for more information).

	Port 1		Port 2		Port 3		Port 4	
Frequency [THz]	Att. [dB]	Phase [rad]	Att. [dB]	Phase [rad]	Att. [dB]	Phase [rad]	Att. [dB]	Phase [rad]
191.250	0.000	0.000	0.500	0.000	60.000	0.000	6.024	3.142
191.252	0.000	0.000	0.500	0.000	27.080	3.142	5.511	3.142
…	…	…	…	…	…	…	…	…
196.274	5.756	0.000	0.500	0.000	6.305	3.142	3.016	3.142
196.275	6.020	0.000	0.500	0.000	6.024	3.142	3.012	3.142

*Table 6.8 Sample PSP data. Note that the headings are for instruction only and are not part of the *.psp file.*

Number of Frequency Data Points

Unlike **.ucf* files, every GHz must be specified for the whole available spectrum in every **.psp* file. The first and last points in the file must be as per specification for the respective WaveShaper model. To avoid interpolation and rounding problems, it is recommended to check that the frequencies are specified to at least 3 decimal places in the **.psp* file.

Frequency

Each frequency data point (specified in absolute terms for **.psp* files) must have a corresponding value of Attenuation, Phase. If no specific attenuation or phase value is required, values of 0 must be specified at these points.

Attenuation

The calibrated attenuation values available in the WaveShaper hardware are 0 to 30 dB. The calculation of the attenuation levels in the Fourier Processor software requires an iterative method however. This method will try to find the closest attenuation and phase levels to the ones specified in the **.psp* file, hence there is no guarantee of accuracy when using the Fourier Processor software.

Phase

The phase control range available in the WaveShaper is 0-2π. The **.psp* file may specify a phase outside of this range, however, this will be re-calculated by the WaveShaper software on interpolation as (phase modulo 2π) as shown in Figure 6.1

6.5.3 (Optional) Header

It is possible to include optional header items at the top of the PSP file. They provide additional information about the PSP. Each header item should have its own line, and be prefixed with a hash '#' character.

The Port Allocation Vector Header Item

The port allocation vector allows power splitting on less than 4 ports. It consists of a comma-delimited list of port numbers mapping the active ports to their associated attenuation and phase column pair in the body. I.e. the n^{th} port number in the vector maps attenuation and phase to columns $2n$ and $2n+1$ respectively in the PSP body. The number of columns expected in the PSP body is then $1+2N$ when there are N ports in the port allocation vector.

Consider this example port allocation header line:

#4,2

This indicates the following information:

- Only Ports 2 and 4 are activated. The behaviour at Ports 1 and 3 is undefined.
- The attenuation and phase information for Port 4 is given by columns 2 and 3 respectively.
- The attenuation and phase information for Port 2 is given by columns 4 and 5 respectively.
- The main PSP body should only contain 5 columns.

When power splitting is performed on less than 4 ports, there are advantages to using the port allocation vector compared to using dummy values for the unused ports. These advantages include increased optical transmission, reduced *.psp file size, and increased profile generation speed.

Unscaled Mode Header Item

By default, a PSP profile is set to "scaled" mode. Use the following header item to switch to Unscaled mode:

#Unscaled

Scaled mode

All ports are allocated an equal share of optical power. An attenuation of zero corresponds to a port transmitting its entire share. It is not possible for a given port to transmit more power than its given share. E.g. it is not possible to transmit ALL of the optical power to a single port when power splitting with 4 active ports under scaled mode. Equivalently, "Scaled" mode introduces the following attenuation to each port:

Number of Active Ports	Additional Attenuation [dB]
1	0
2	3
3	4.8
4	6

Unscaled mode

The attenuation value specified for the port is the amount of the available input optical power transmitted to the port. Note: clipping can occur in this mode if total sum of requested power levels on any wavelength is larger than 100% of the incoming light.

7.1 Introduction

7.2 Installation

7.3 User Instructions

Section 7: WaveShaper WSUTIL User Guide

7.1 Introduction

WSUTIL is a console program for computing and loading filter profiles on to a WaveShaper device. It generates the WaveShaper profile from a WaveShaper configuration file (.wsconfig) and a WaveShaper Preset file (.wsp). Please see Section 6: WaveShaper File Formats for descriptions of user generated input files. The wsconfig file is unique for each WaveShaper unit and is supplied on the installation USB Drive or CD shipped with the unit. This software automatically initializes the WaveShaper and then loads the generated profile. This software also supports firmware updates.

7.2 Installation

The WaveShaper command line software is automatically installed upon installation of the WaveManager package.

7.3 User Instructions

Connect the WaveShaper to the computer's USB port and then switch on power to the WaveShaper unit. Note that WaveShaper device needs about 30 seconds to complete initialization after power up. From the Start menu Run the Finisar WSUTIL program.

The wsutil commands can now be run from the command prompt.

7.3.1 Help

To obtain a list of wsutil commands, enter the following:

```
wsutil -h
```

A sample of the WSUTIL -h command result is given below:

```
wsutil version:1.5.8
Usage: wsutil <wsconfig> [OPTION] ..
<wsconfig> waveshaper config file (e.g. sn009070.wsconfig) - This is a
required parameter for all actions
-l, --loadprofile <wspfile> load profile(accepted file type: *.wsp)
-f, --firmware <binfile> upload firmware(accepted file type: *.bin)
--list list serial numbers of all connected
WaveShapers
WSUTIL User Guide
Revision C Page 6 of 7
--sno <sno> specify serial number for wconfig and
rconfig
--rconfig <wsconfig> read config file from embedded flash memory
--wconfig <wsconfig> write config file to embedded flash memory
--cfg <config> select sub-configuration part
-h, --help print this help information
Example: Upload filter profile to waveshaper.
wsutil sn009070.wsconfig -l test.wsp
```

7.3.2 Load WaveShaper Profile

To compute and load a WaveShaper profile, the user needs to specify the WaveShaper configuration file (.wsconfig) and .wsp file. The command format is as below:

```
wsutil <wsconfig> -l <wsp>
```

Example:

```
wsutil snxxxxx.wsconfig –l test.wsp
```

7.3.3 Select Sub-configuration (WaveShaper 2000 only)

This option selects the sub-configuration part on WaveShaper with multiple configuration regions. The command format is as below:

```
wsutil <wsconfig> ---cfg <firmware file>
```

Example:

```
wsutil <wsconfig> --cfg w0 -l test.wsp   //for uploading to the X-polarization
wsutil <wsconfig> --cfg w1 -l test.wsp   //for uploading to the Y-polarization.
```

7.3.4 Read Firmware Version

This option reads WaveShaper device firmware version. The command format is as below:

```
wsutil <wsconfig> --fwver
```

7.3.5 Update Firmware

To download and upgrade the firmware, the user needs to specify the WaveShaper configuration file (.wsconfig) and firmware file (.bin file). Please only upload new firmware to the WaveShaper when specifically instructed to do so by a Finisar representative.

```
wsutil <wsconfig> -f <firmware file>
```

Example:

```
wsutil snxxxxx.wsconfig -f firmware.bin
```

7.3.7 List Connected WaveShapers

To obtain a list of powered up and connected WaveShapers, enter the following:

```
wsutil --list
```

Example:

```
wsutil --list
```

Returns:

```
'53841;15020;9070'
```

7.3.8 Read Configuration File

To read the embedded configuration file from a WaveShaper flash memory, the user needs to specify the serial number of the WaveShaper Device and the file location and name (.wsconfig) under which the configuration file is to saved. This is only available for WaveShaper Firmware versions 2.2.19 or later.

```
wsutil --sno <serial Number> --rconfig <.wsconfig>
```

Example:

```
wsutil --sno 9070 --rconfig  C:\Finisar\wsconfig\sn009070.wsconfig
```

Returns:

```
crc verification passed
size of configuration is 4308196
crc verification passed
read section size = 262144
checksum verified
read section size = 262144
checksum verified
read section size = 262144
checksum verified
read section size = 262144
checksum verified
read section size = 262144
checksum verified
read section size = 68376
checksum verified
File decompressed size = 4308196
```

7.3.9 Embed Configuration Files

To write a configuration file to WaveShaper flash memory, the user needs to specify the serial number of the WaveShaper Device and the file name (.wsconfig) from which the configuration file is copied. This is only available for WaveShaper Firmware versions 2.2.19 or later.

```
wsutil --sno <serial Number> --wconfig <.wsconfig>
```

Example:

```
wsutil --sno 9070 --wconfig  sn009070.wsconfig
```

Returns:

```
File compressed size = 4308196 compressed = 1379096
section written 262144, crc=2f6c3813
section written 262144, crc=b6c409a9
section written 262144, crc=34faf15
section written 262144, crc=3bbcc814
section written 262144, crc=d624a476
section written 3096, crc=2f6c3813
header written 3096
Write configuration success
```

8.1 Overview

8.2 Development Environment Setup

8.3 WS Command Overview and Program Structure

8.4 WS API Functions
- *WS_CREATE_WAVESHAPER*
- *WS_CREATE_WAVESHAPER4*
- *WS_OPEN_WAVESHAPER*
- *WS_CLOSE_WAVESHAPER*
- *WS_LOAD_PROFILE*
- *WS_LOAD_PREDEFINEDPROFILE*
- *WS_GET_RESULT_DESCRIPTION*
- *WS_GET_SNO*
- *WS_GET_FREQUENCYRANGE*
- *WS_GET_PORTCOUNT*
- *WS_GET_PROFILE*
- *WS_GET_VERSION*
- *WS_GET_CONFIGVERSION*
- *WS_LOAD_FIRMWARE*
- *WS_LIST_DEVICES*
- *WS_CREATE_WAVESHAPER_FROMSNO*
- *WS_READ_CONFIGDATA*
- *WS_WRITE_CONFIGDATA*
- *WS_LOAD_PROFILE_FOR_MODELING*
- *WS_GET_MODEL_PROFILE*
- *WS_SEND_COMMAND*
- *WS_CREATE_WAVESHAPER_FORSIMULATION*

8.5 PS Command Overview and Program Structure

8.6 PS API Functions
- *PS_CREATE_PSOBJECT*
- *PS_DELETE_PSOBJECT*
- *PS_OPEN_WAVESHAPER*
- *PS_CLOSE_WAVESHAPER*
- *PS_LOAD_PSP*
- *PS_LOAD_PREDEFINEDPROFILE*
- *PS_GET_FREQUENCYRANGE*

8.7 Error Code Type Definitions

Section 8: WaveShaper API Programming Guide

8.1 Overview

The WaveShaper "WS" API functions provide the ability to load a WSP (WaveShaper Preset) input, compute the resulting filter profile and, subsequently, download the filter/switch profile to a WaveShaper device. Alternatively, if the WaveShaper is used in power splitting mode , the "PS" API functions provide the ability to load a PSP (WaveShaper Power Split Preset) input, compute the resulting profile and, subsequently, download the profile to a WaveShaper device. These are a separate set of API commands and cannot be used in conjunction with any commands prefixed with 'ws', with the exception of ws_read_configdata, ws_list_devices, ws_get_version and ws_get_result_description. As the WaveShaper Object and the Powersplitting Object are fundamentally different, they cannot be used together.

The API is available for Linux and Windows operating systems. The WSP and PSP file formats used by the API are detailed in Section 6: WaveShaper File Formats.

This section describes the structure of a program to control a WaveShaper using the API with programming examples in C and Python and provides a complete list of all the commands, together with sample C code for each function.

In order to access the latest functions, it is recommended to upgrade the WaveShaper firmware (eg to ws25G-02_02_19.bin). Please see section 7.3.4 on page 46

8.2 Development Environment Setup

Customized WaveShaper applications can be developed with different programming languages. C and Python API are provided. If another programming language is used, the C API will have to be wrapped in that language.

All files necessary for the development environment are installed when the WaveManager Application Suite is installed (see Section 2: Getting Started).

8.2.1 Using the WaveShaper API Files with C

The WaveShaper API includes the following files. After the WaveShaper software is installed, they can be found in the installation directory indicated in the table below.

File Name	Description
waveshaper\api\include\ws_api.h	Header file contains all function and type definitions.
waveshaper\api\include\ws_psapi.h	Optional Header file containing all Power Splitting functions
waveshaper\api\wsapi.dll (.so)	WaveShaper API dynamic link library.

File Name	Description
waveshaper\api\wsapi.lib	Static import library to wsapi.dll (Windows only)

Table 8.1 C API Files

The header file ws_api.h should be included in the WaveShaper API C code . If the WaveShaper is being run in power splitting mode, the ws_psapi.h must also be included. The static import library is used by the linker to resolve WaveShaper API symbols during linking process.

The following table lists the dependencies of wsapi.dll. All of them must be placed in a directory that is included in the system's dynamic library loading path.

File Name	Description
FTD2XX.DLL	FTDI Serial API (Windows only, needs to be in Windows\System32 directory).
ws_cheetah.DLL (ws_cheetah.so)	Cheetah SPI API, needs to be in the same directory as wsapi.dll

Table 8.2 WaveShaper API Dependencies

For Linux, these files should be put into standard library directory (e.g. /usr/lib)

8.2.2 Using the WaveShaper API with Python

Python API is provided to support accessing both "WS" and "PS" WaveShaper functions from Python scripts. The module setup script is located in waveshaper\api\ python sub-directory. The "*setup.py*" script is provided to install WaveShaper API python module into user's python environment. An example of invoking the setup script is shown below.

The file wsapi.dll and all DLL dependencies (FTD2XX.DLL(so) and ws_cheetah.DLL) must be placed in a directory that is included in the system's dynamic library loading path, so that the user's python script can access WaveShaper API functions.

1. Make sure both Python and WaveManager are installed.
2. From the command line, navigate to the following directory using the command line:
   For Windows XP: *C:\Program Files\WaveManager\waveshaper\api\python*
   For Windows 7: *C:\Program Files (x86)\WaveManager\waveshaper\api\python*

   This directory may change depending on where WaveManager has been installed.
3. Once in this directory, run the setup.py by entering:

   python.py install

   This is necessary to allow Python to access the API commands.

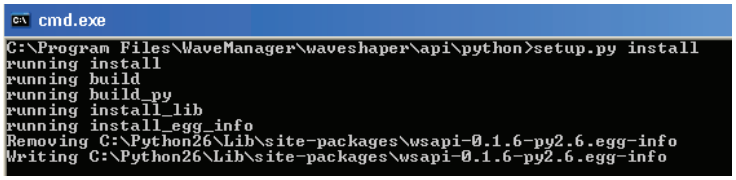


Figure 8.1 Setup.py installation script preparing the WaveShaper environment in Python.

4. The wsapi.dll and dependancies FTD2XX.DLL(so) and ws_cheetah.DLL files can be found in the following directory:
   For Windows XP: *C:\Program Files\WaveManager\waveshaper\api*
   For Windows 7: *C:\Program Files (x86)\WaveManager\waveshaper\api*

   They can be copied into the current working directory as your **.py* file to automatically be included int the system's dynamic library loading path when starting the pythion program from within this directory.

5. The API commands can now be imported and used by including the following header line in your python code:

```
from wsapi import *
```

8.3 WS Command Overview and Program Structure

Before using any WaveShaper API commands, the application must first create a WaveShaper instance and, at the application termination sequence, it must close and delete the WaveShaper instance. The process overview is shown in flow chart below.

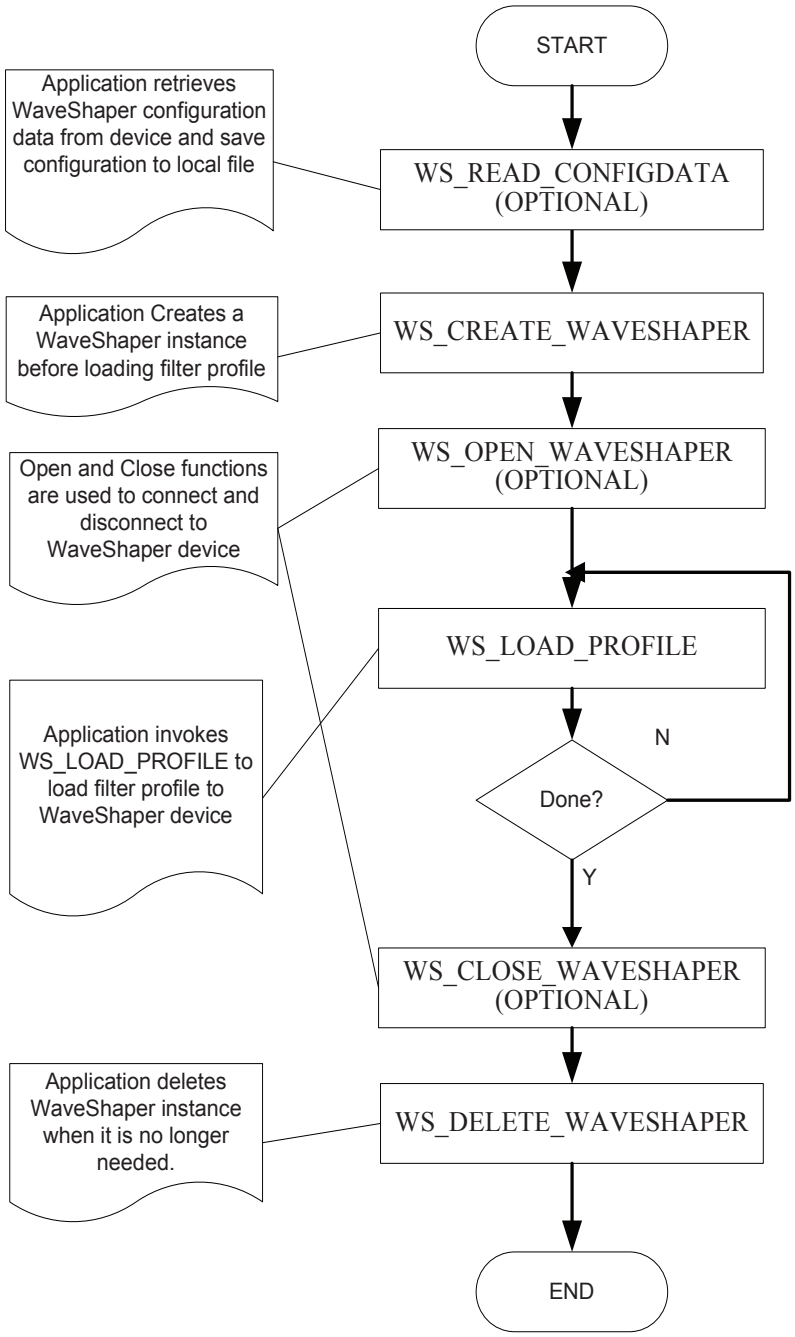


Figure 8.2 Program Structure Flow Chart

WaveShapers with updated firmware supports configuration data embedding. The WaveShaper configuration data can be downloaded from the device by following the sequence below.

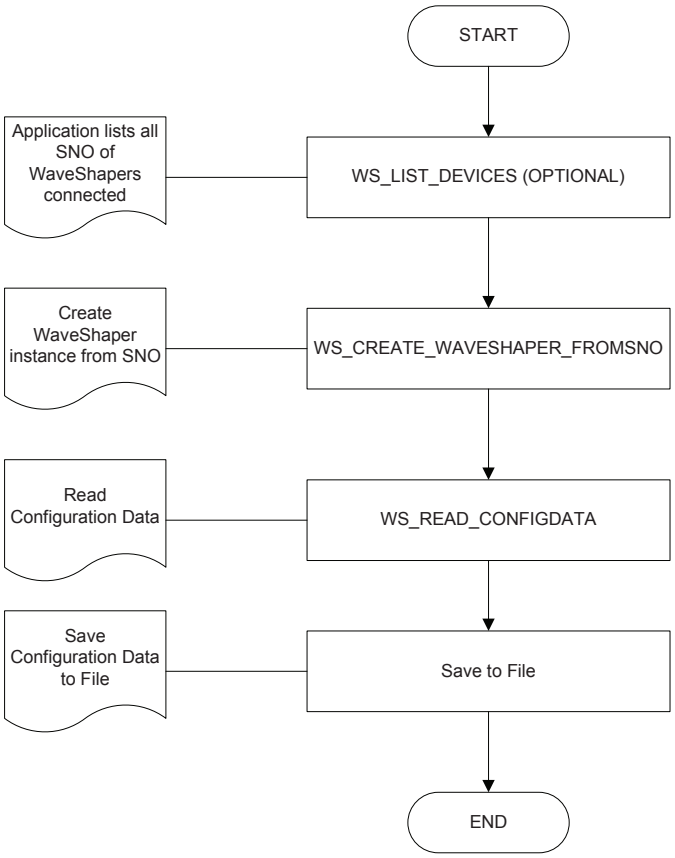


Figure 8.3 read_config_data Flow Chart

8.3.1 C Sample Code

```
/** @file simple.cpp
 * @author Finisar Australia - Copyright (c) 2005-2011
 *
 * This is a sample program which shows how to load a filter profile
 * through ws_load_profile function.
 * It also demonstrates how to initialize the API and create the
 * WaveShaper instance.
 */

#include <string.h>
#include <stdlib.h>
#include "ws_api.h"

//profile text buffer
static char profiletext[1024 * 1024];

int main(int argc, char** argv) {
      int rc=0;

      //check input number
      if(argc != 3) {
        printf("Error: 2 arguments expected: [wsconfig] [wsp]\n");
        printf("Example: %s SN007090.wsconfig test.profile\n", argv[0]);
        return rc;
      }


      //create a waveshaper object and name it 'ws1'
      rc = ws_create_waveshaper("ws1", argv[1]);
      if(rc !=WS_SUCCESS ) {
        printf("Create WaveShaper Error: %s\n", ws_get_result_description(rc));
        return rc;
      }
      printf("Create WaveShaper OK\n");

      //open WSP file and load profile text
      FILE* fp = fopen( argv[2], "r");
      if(fp == NULL) {
        printf("Error: can not open file %s\n", argv[2]);
        return -1;
      }
      rc = fread(profiletext, 1, sizeof(profiletext)-1, fp);
      profiletext[rc] = '\0';

      //load WSP profile from profile text
      rc = ws_load_profile("ws1", profiletext);
      if(rc !=WS_SUCCESS ) {
        printf("Load Profile Error: %s\n", ws_get_result_description(rc));
        return rc;
      }
      printf("Load Profile OK\n");

      //delete waveshaper object
      rc = ws_delete_waveshaper("ws1");
      if(rc !=WS_SUCCESS ) {
        printf("Delete WaveShaper Error: %s\n", ws_get_result_description(rc));
        return rc;
      }


      printf("Load Profile Done\n");
      return 0;
}
```

8.3.2 Python Sample Code

```
#import wsapi python wrapper
from wsapi import *

#create waveshaper instance and name it "ws1"
rc = ws_create_waveshaper("ws1", "testdata/SN007090.wsconfig")
print "ws_create_waveshaper rc="+ws_get_result_description(rc)

#read profile from WSP file
WSPfile = open('testdata/test 100GHz 4ports alternating.wsp', 'r')
profiletext = WSPfile.read()

#compute filter profile from profile text, then load to Waveshaper device
rc = ws_load_profile("ws1", profiletext)
print "ws_load_profile rc="+ws_get_result_description(rc)

#delete the waveshaper instance
rc = ws_delete_waveshaper("ws1")
print "ws_delete_waveshaper rc="+ws_get_result_description(rc)
```

```
#import wsapi python wrapper
from wsapi import *
import matplotlib.pyplot as plt
import numpy as np

#Create the WSP vector Data
wsFreq = np.arange(191.25, 196.275, 0.001)
wsAttn = 10*np.sin(50*(wsFreq-191.25)/2/np.pi)+10
wsPhase = 2*np.pi*np.cos(5*(wsFreq-191.25))
wsPort = np.ones(np.size(wsFreq))

#Create the WSP file
WSPfile = open('TrigProfile.wsp', 'w')
for x in range(np.size(wsFreq)):
    WSPfile.write("%0.3f\t%0.3f\t%0.3f\t%0.3f\n" % (wsFreq[x], wsAttn[x],
wsPhase[x], wsPort[x]))
WSPfile.close()

#Read Profile from WSP file
WSPfile = open('TrigProfile.wsp', 'r')
profiletext = WSPfile.read()
WSPfile.close()

#create waveshaper instance and name it "ws1"
rc = ws_create_waveshaper("ws1", "SN024187.wsconfig")
print "ws_create_waveshaper rc="+ws_get_result_description(rc)

#compute filter profile from profile text, then load to Waveshaper device
rc = ws_load_profile("ws1", profiletext)
print "ws_load_profile rc="+ws_get_result_description(rc)

#delete the waveshaper instance
rc = ws_delete_waveshaper("ws1")
print "ws_delete_waveshaper rc="+ws_get_result_description(rc)
```

8.4 WS API Functions

8.4.1 WS_CREATE_WAVESHAPER

int ws_create_waveshaper (const char * name, const char * wsconfig)

Creates a WaveShaper object instance with a user specified name. The user can choose any name containing one or more letters or digits or underscore, provided that distinct names are used for each WaveShaper device.

Note that WS_CREATE_WAVESHAPER object will not open the communication port or make connections to the WaveShaper device. Opening a connection to the device is done by invoking the WS_OPEN_WAVESHAPER function.

Parameters:

name	[in]	User specified WaveShaper name A valid name consists of one or more letters and can contain digits and underscores. Each Waveshaper object must have a unique name
wsconfig	[in]	Path (to Configuration files)

Results:

Result code, WS_SUCCESS if success, otherwise it is an error code.

Result Code	Description
WS_SUCCESS	Command successfully executed
WS_DUPLICATE_NAME	WaveShaper name already in use

Example : Create WaveShaper instance without call back function

```
int rc = ws_create_waveshaper("ws1", "sn007090.wsconfig");
if (rc == WS_SUCCESS)
      ; WaveShaper object created
else
      ; Error handling code
```

8.4.2 WS_CREATE_WAVESHAPER4

int ws_create_waveshaper4(const char * name, const char * wsconfig , const char* cfg)

Creates a WaveShaper 2000S object instance with a user specified name and sub-configuration part.

It is similar to WS_CREATE_WAVESHAPER where the cfg parameter selects the sub-configuration part on WaveShaper with multiple configuration regions. (“w0” corresponding to one polarization, “w1” to the other polarization).

Parameters:

name	[in]	User specified WaveShaper name A valid name consists of one or more letters and can contain digits and underscores. Each Waveshaper object must have a unique name
wsconfig	[in]	Path string to Configuration file
cgg	[in]	Select sub-configuration part

Results:

Result code, WS_SUCCESS if success, otherwise it is an error code.

Result Code	Description
WS_SUCCESS	Command successfully executed
WS_DUPLICATE_NAME	WaveShaper name already in use

Example : Create WaveShaper instance with sub-configuration part

```
int rc = ws_create_waveshaper4("ws1", "sn007090.wsconfig", "w0");
if (rc == WS_SUCCESS)
      ; WaveShaper object created
else
      ; Error handling code
```

8.4.3 WS_DELETE_WAVESHAPER

int ws_delete_waveshaper (const char * name)

This command deletes the WaveShaper object. The WaveShaper object will be automatically closed, if it is in open state.

Parameters:

name	[in]	Previously created WaveShaper name

Results:

Result code, WS_SUCCESS if success, otherwise it is an error code.

Result Code	Description
WS_SUCCESS	Command successfully executed
WS_WAVESHAPER_NOT_FOUND	WaveShaper object does not exist

Example :

```
int rc;
rc = ws_create_waveshaper("ws1","sn007090.wsconfig", "w0");
...
rc = ws_delete_waveshaper("ws1");
```

8.4.4 WS_OPEN_WAVESHAPER

int ws_open_waveshaper (const char * name)

This command opens an existing WaveShaper object and establishes the connection to the defined WaveShaper.

Note that it is optional to call this function before invoking profile downloading functions. WaveShaper API will automatically open the connection before downloading profile if it is not opened yet.

However, it is recommended to invoke “ws_open_waveshaper” explicitly for better performance, when downloading multiple filter profiles. Downloading speed will be increased, as no re-connection is needed between downloading each new profile.

Parameters:

name	[in]	User specified WaveShaper name

Results:

Result code, WS_SUCCESS if success, otherwise it is an error code.

Result Code	Description
WS_SUCCESS	Command successfully executed
WS_WAVESHAPER_NOT_FOUND	WaveShaper object does not exist
WS_OPENFAILED	Could not open the WaveShaper. Possible connection problem.

Example :

```
int rc = ws_open_waveshaper("ws1");
if (rc == WS_SUCCESS)
      ; Connection to WaveShaper successfully established
else
      ; Error handling code
```

8.4.5 WS_CLOSE_WAVESHAPER

int ws_close_waveshaper (const char * name)

Close WaveShaper. Disconnect from the WaveShaper device.

Note that it is optional to call this function. The connection will be automatically closed during deletion of WaveShaper object (see WS_DELETE_WAVESHAPER).

Parameters:

name	[in]	Previously created WaveShaper name

Results:

Result code, WS_SUCCESS if success, otherwise it is an error code.

Result Code	Description
WS_SUCCESS	Command successfully executed
WS_WAVESHAPER_NOT_FOUND	WaveShaper object does not exist

Example :

```
iint rc = ws_close_waveshaper("ws1");
if (rc == WS_SUCCESS)
      ; Connection to WaveShaper successfully closed
else
      ; Error handling code
```

8.4.6 WS_LOAD_PROFILE

int ws_load_profile (const char * name, const char * profiletext)

Apply WSP filter and wait for completion. Calculate the filter profile based on WSP-text, then load filter profile to WaveShaper device. .

Parameters:

name	[in]	Previously created WaveShaper name
profiletext	[in]	WSP text string

Results:

Result code, WS_SUCCESS if success, otherwise it is an error code.

Result Code	Description
WS_SUCCESS	Command successfully executed
WS_WAVESHAPER_NOT_FOUND	WaveShaper object does not exise
WS_INVALIDPORT	Port number is not valid
WS_INVALIDFREQ	Frequency specified out of range
WS_INVALIDATTN	Attenuation is not valid (e.g. negative value)
WS_INVALIDSPACING	Frequencies not incremented in 0.001 THz step
WS_NARROWBANDWIDTH	Bandwidth of frequencies to the same port is less than 0.010 THz
WS_INVALIDPROFILE	Other parsing error
WS_OPENFAILED	Could not open the WaveShaper. Possible connection problem.
WS_WAVESHAPER_CMD_ERROR	Error response from WaveShaper. May be communication corruption.

Example :

```
/* hard coded profile text */
char* profiletext="192.000 1.0 0.0 1\n192.001 1.0 0.0 1\n..";
int rc = ws_load_profile("ws1", profiletext);
if (rc == WS_SUCCESS)
      ; Profile loaded
else
      ; Error handling code
```

8.4.7 WS_LOAD_PREDEFINEDPROFILE

int ws_load_predefinedprofile(const char* name, int filtertype, float center, float bandwidth, float attn, int port)

This function allows the user to apply one of several pre-defined filters. The spectral profile is calculated based on the input parameters, and then is uploaded to the target WaveShaper, waiting for the operation to complete.

Parameters:

name	[in]	Previously created WaveShaper name
filtertype	[in]	Predefine filter type (see filter type table below)
center	[in]	Center Frequency (THz). Set to 0.0f, if not used.
bandwidth	[in]	Bandwidth (GHz) . Set to 0.0f, if not used.
attn	[in]	Attenuation (dB) . Set to 0.0f, if not used. Attenuation must be a positive number in the range 0 to 40 dB.
port	[in]	Port number. Set to 0, if not used

Filter Type List:

Filter Type	Description
PROFILE_TYPE_BLOCKALL	Block the entire optical spectrum of the WaveShaper. Used parameters: type

Filter Type	Description
PROFILE_TYPE_TRANSMIT	Transmit the entire optical spectrum to the desired output port. Used parameters: type, port
PROFILE_TYPE_BANDPASS	Band pass filter. Used parameters: type, center, bandwidth, attn, port
PROFILE_TYPE_BANDSTOP	Band stop filter. Used parameters: type, center, bandwidth, port
PROFILE_TYPE_GAUSSIAN	Gaussian filter. Used parameters: type, center, bandwidth, attn, port

Returns:

Result Code	Description
WS_SUCCESS	Command successfully executed
WS_WAVESHAPER_NOT_FOUND	WaveShaper object does not exist
WS_INVALIDPORT	Port number is not valid
WS_INVALIDFREQ	Frequency specified out of range
WS_INVALIDATTN	Attenuation is not valid (e.g. negative value)
WS_NARROWBANDWIDTH	Bandwidth of frequencies to the same port is less than 0.010 THz
WS_INVALIDPROFILE	Other parsing error
WS_OPENFAILED	Could not open the WaveShaper. Possible connection problem.
WS_WAVESHAPER_CMD_ERROR	Error response from WaveShaper. Possible communication corruption.

Example :

```
/* block all*/
int rc = ws_load_predefinedprofile ("ws1",
 PROFILE_TYPE_BLOCKALL, 0.0f, 0.0f, 0.0f, 0);
if (rc == WS_SUCCESS)
      ; Profile loaded
else
      ; Error handling code
```

8.4.8 WS_GET_RESULT_DESCRIPTION

const char* ws_get_result_description (int rc)

Get text description from result code.

Parameters:

rc	[in]	Result code

Returns:

Text description of the result code .

Result Code	Description
WS_SUCCESS	Command successfully executed
WS_WAVESHAPER_NOT_FOUND	WaveShaper object does not exist

Example :

```
int rc = ws_create_waveshaper("ws1", "sn1234.wsconfig");
printf("Create WaveShaper Result: %s\n",
       ws_get_result_description(rc));
```

8.4.9 WS_GET_SNO

```
int ws_get_sno(const char* name, char* sno, int size)
```

Get WaveShaper serial number. Can be used to confirm which WaveShaper is connected in a system where multiple WaveShapers are connected through some form of switched interface.

Parameters:

name	[in]	Previously created WaveShaper name
sno	[in]	Buffer to hold serial number
size	[in]	Buffer size

Returns:

Result code, WS_SUCCESS if success, otherwise it is an error code.

Result Code	Description
WS_SUCCESS	Command successfully executed
WS_WAVESHAPER_NOT_FOUND	WaveShaper object does not exist

Example :

```
char buffer[64];
int rc = ws_get_sno ("ws1", buffer, 64);
printf("Serial number is %s\n", buffer);
```

8.4.10 WS_GET_FREQUENCYRANGE

```
int ws_get_frequencyrange(const char* name, float* start, float* stop)
```

Get start and stop frequency of the WaveShaper. This allows the user to determine, for instance, if the attached WaveShaper is for C- or L-band operation. This is also a useful check to ensure that a WSP does not extend beyond the operating range of the unit being controlled.

Parameters:

name	[in]	Previously created WaveShaper name
start	[in]	Starting Frequency (THz)
stop	[in]	Stopping Frequency (THz)

Returns:

Result code, WS_SUCCESS if success, otherwise it is an error code.

Result Code	Description
WS_SUCCESS	Command successfully executed
WS_WAVESHAPER_NOT_FOUND	WaveShaper object does not exist

Example :

```
float start, stop;
int rc = ws_get_frequencyrange ("ws1", &start, &stop);
printf("Frequency range: %f-%f\n", start, stop);
```

8.4.11 WS_GET_PORTCOUNT

```
int ws_get_portcount(const char* name, int* nport)
```

Get number of ports supported by the WaveShaper. This can be used to determine if the device is a WaveShaper 1000S, WaveShaper 4000S or WaveShaper 16000S. For each device, the nuber of ports, nport, is the number of unique input-output port combinations that exist within that device. So, for a 1000S, nport = 1; for a 4000S, nport = 4 and for a 16000S, nport = 64 (4 x 16).

Parameters:

name	[in]	Previously created WaveShaper name
nport	[out]	Port count

Returns:

Result code, WS_SUCCESS if success, otherwise it is an error code.

Result Code	Description
WS_SUCCESS	Command successfully executed
WS_WAVESHAPER_NOT_FOUND	WaveShaper object does not exist

Example :

```
int nport;
int rc = ws_get_portcount ("ws1", &nport);
printf("Port count: %d\n", nport);
```

8.4.12 WS_GET_PROFILE

```
int ws_get_profile(const char* name, char* profilebuffer, int* psize)
```

Get WSP representation of currently loaded filter profile. This function can be used to determine WSP version of a currently-loaded filter. This can be useful when a series of partial WSP files have been loaded and the user wishes to analyze or save the current status of the WaveShaper.

psize points to the size of the buffer variable as input. Upon completion, the size variable will be updated with actual WSP text length. If the output size is larger than input size, user should re-allocate a larger buffer and invoke the function again to get full WSP text.

Parameters:

name	[in]	Previously created WaveShaper name
profilebuffer	[out]	Buffer to hold the output WSP text
psize	[in/out]	Size of the buffer, and size of output

Returns:

Result code, WS_SUCCESS if success, otherwise it is an error code.

Result Code	Description
WS_SUCCESS	Command successfully executed
WS_WAVESHAPER_NOT_FOUND	WaveShaper object does not exist

Example :

```
char* buffer = malloc(500000);
int size = 500000;
int rc = ws_get_profile("ws1", buffer, &size);
printf("profile: %s\n", buffer);
free(buffer);
```

8.4.13 WS_GET_VERSION

```
const char* ws_get_version()
```

Get WaveShaper DLL version..

Returns:

NULL terminated version string in [major].[minor].[build] format (e.g. 1.0.10).

Example :

```
const char* version = ws_get_version ();

printf("DLL version is: %s\n", buffer);
```

8.4.14 WS_GET_CONFIGVERSION

```
int ws_get_configversion(const char* name, char* version)
```

Get WaveShaper configuration version. Version is coded in [major].[minor] format (e.g. 1.2).

Parameters:

name	[in]	Previously created WaveShaper name
version	[out]	Buffer to hold the output version string. Buffer size must be at least 32 bytes long.

Returns:

Result code, WS_SUCCESS if success, otherwise it is an error code.

Result Code	Description
WS_SUCCESS	Command successfully executed
WS_WAVESHAPER_NOT_FOUND	WaveShaper object does not exist

Example :

```
char buffer[32];
int rc = ws_get_configversion ("ws1", buffer);
printf("WaveShaper configuration version is: %s\n", buffer);
```

8.4.15 WS_LOAD_FIRMWARE

```
int ws_load_firmware(const char* name, const char* filename, char* oldver, char* newver)
```

Load new version of firmware to WaveShaper device.

Parameters:

name	[in]	Previously created WaveShaper name
filename	[in]	Path to firmware file
oldver	[out]	Pointer to old firmware version buffer, must be larger than 64bytes
newver	[out]	Pointer to old firmware version buffer, must be larger than 64bytes

Returns:

Result code, WS_SUCCESS if success, otherwise it is an error code.

Result Code	Description
WS_SUCCESS	Command successfully executed
WS_WAVESHAPER_NOT_FOUND	WaveShaper object does not exist

Example :

```
char buffer[32];
char oldver[64], newver[64];
int rc = ws_load_firmware ("ws1", "newfirmware.bin", oldver, newver);
if (rc == WS_SUCCESS)
      ; Firmware updated
else
      ; Error handling code
```

8.4.16 WS_LIST_DEVICES

```
int ws_list_devices(char* buffer, int buffersize);
```

List the serial numbers of all connected devices.

Note that this function may not be able to enumerate old WaveShaper devices.

Parameters:

buffer	[in]	Buffer to receive serial numbers of connected devices
buffersize	[in]	Buffer Size

Returns:

Result code, WS_SUCCESS if success, otherwise it is an error code.

Example :

```
char buffer[128];
memset(buffer, 0, sizeof(buffer));
int  rc = ws_list_devices (buffer,sizeof(buffer));
if (rc == WS_SUCCESS)
      printf("WaveShaper connected: %s", buffer);
else
      ; Error handling code
```

8.4.17 WS_CREATE_WAVESHAPER_FROMSNO

```
int ws_create_waveshaper_fromsno(const char* name, const char* sno);
```

This function creates a named WaveShaper object instance for the device serial number. The user can choose any name containing one or more letters or digits or underscore, provided that distinct names are used for each WaveShaper device.

This function is used to create a WaveShaper instance in order to read configuration data from device.

Note that the WaveShaper object created by use of this function can not be used to load a filter profile, because calibration data is not loaded.

Parameters:

name	[in]	User specified WaveShaper name. A valid name contains one or more letters or digits or underscore. Each WaveShaper object must have a unique name.
sno	[in]	Serial number

Returns:

Result code, WS_SUCCESS if success, otherwise it is an error code.

Result Code	Description
WS_SUCCESS	Command successfully executed
WS_WAVESHAPER_NOT_FOUND	WaveShaper object does not exist
WS_OPENFAILED	Could not open the WaveShaper. May be a connection problem.

Example :

```
int  rc = ws_create_waveshaper_fromsno ("ws1", "1234");
printf("Create WaveShaper Result: %s\n",
         ws_get_result_description(rc));
```

8.4.18 WS_READ_CONFIGDATA

```
int ws_read_configdata(const char* name, char* buffer, int buffersize, int* nread);
```

Read configuration data from WaveShaper flash memory.

Parameters:

name	[in]	Previously created WaveShaper name (see ws_create_waveshaper_fromsno))
buffer	[in]	Buffer to receive embedded data read from device Set to NULL, to get the configuration data size (output to nread)
buffersize	[in]	Buffer Size
nread	[out]	Number of bytes read

Returns:

Result code, WS_SUCCESS if success, otherwise it is an error code.

Example :

```
char buffer[10000000]
int nread =0;
int  rc = ws_read_configdata("ws1", buffer, sizeof(buffer), &nread);
if (rc == WS_SUCCESS)
      ; Save data in buffer to file
else
      ; Error handling code
```

8.4.19 WS_WRITE_CONFIGDATA

```
int ws_write_configdata(const char* name, char* buffer, int size, int* nwrite);
```

Write configuration data to WaveShaper flash memory.

Parameters:

name	[in]	Previously created WaveShaper name (see ws_create_waveshaper_fromsno)
buffer	[in]	Buffer holding embedded data to write to device
size	[in]	Buffer Size
nwrite	[out]	Number of bytes written

Returns:

Result code, WS_SUCCESS if success, otherwise it is an error code.

Example :

```
char buffer[10000000]
int nwrite =0;
int   rc = ws_write_configdata("ws1", buffer, sizeof(buffer), &nwrite);
if (rc == WS_SUCCESS)
      ; returns no bytes written
else
      ; Error handling code
```

8.4.20 WS_LOAD_PROFILE_FOR_MODELING

```
int ws_load_profile_for_modeling(const char* name, const char* wsptext, int port,
void* resv);
```

Load WSP filter to internal buffer for modeling. Calculate the simulated filter profile based on WSP-text

Parameters:

name	[in]	Previously created WaveShaper name.
profiletext	[in]	WSP text string.
port	[in]	Port to be simulated.
resv	[in]	Reserved parameter, set to NULL

Returns:

Result code, WS_SUCCESS if success, otherwise it is an error code.

Result Code	Description
WS_SUCCESS	Command successfully executed
WS_WAVESHAPER_NOT_FOUND	WaveShaper object does not exist
WS_INVALIDPORT	Port number is not valid.
WS_INVALIDFREQ	Frequency specified out of range
WS_INVALIDATTN	Attenuation is not valid (e.g. negative value)
WS_INVALIDSPACING	Frequencies not incremented in 0.001 THz step

Result Code	Description
WS_NARROWBANDWIDTH	bandwidth of frequencies to the same port is less than 0.010 THz
WS_INVALIDPROFILE	Other parsing error

Example :

```
/* hard coded profile text */
char* profiletext="192.000 1.0 0.0 1\n192.001 1.0 0.0 1\n..";
int rc = ws_load_profile_for_modeling ("ws1", profiletext, 1, NULL);
if (rc == WS_SUCCESS)
      ; Profile loaded
else
      ; Error handling code
```

8.4.21 WS_GET_MODEL_PROFILE

int ws_get_model_profile(const char* name, char* wspbuffer, int* psize);

Get WSP representation of currently loaded modeling filter profile (see ws_load_profile_for_modeling).

psize points to the size of the buffer variable as input. Upon completion, the size variable will be updated with actual WSP text length. If the output size is larger than input size, the user should re-allocate a larger buffer and invoke the function again to get the full WSP text.

Parameters:

name	[in]	Previously created WaveShaper name
wspbuffer	[out]	Buffer to hold the output WSP text
psize	[int/out]	Size of the buffer, and size of output

Returns:

Result code, WS_SUCCESS if success, otherwise it is an error code.

Result Code	Description
WS_SUCCESS	Command successfully executed
WS_WAVESHAPER_NOT_FOUND	WaveShaper object does not exist

Example :

```
char* buffer = malloc(500000);
int   size = 500000;
int  rc = ws_get_model_profile ("ws1", buffer, &size);
printf("simulated profile: %s\n", buffer);
free(buffer);
```

8.4.22 WS_SEND_COMMAND

int ws_send_command(const char* name, const char* cmd, char* response, int* responsesize)

Send a raw WaveShaper device level command and receive a response.

Parameters:

name	[in]	Previously created WaveShaper name
cmd	[in]	NULL terminated command string
response	[out]	Response buffer
responsesize	[in | out]	Pointer to the limit of the response buffer size as input, hold the response data length as output

Returns:

Result code, WS_SUCCESS if success, otherwise it is an error code.

Example : Send Command to retrieve case temperature

```
/* Create WaveShaper instance without call back function */
char response[256];
int responsesize = sizeof(response);
memset(response, 0, sizeof(response));; WaveShaper object created
int rc = ws_send_command("ws1", "CSS?\r\n", response,
&responsesize);
if (rc == WS_SUCCESS)
printf("response=%s", (char*)response);
else
; Error handling code
============= expected output ==========
response=CSS?
036.7
OK
```

8.4.23 WS_CREATE_WAVESHAPER_FORSIMULATION

int ws_create_waveshaper_forsimulation(char* name, const char* wsconfig)

Has the same behaviour as WS_CREATE_WAVESHAPER but creates a WaveShaper object for simulation purposes. When used instead of WS_CREATE_WAVESHAPER, all subsequent API functions behave as if the WaveShaper is connected. This enables testing when a WaveShaper is not connected or accessible.

Parameters:

name	[in]	User specified WaveShaper name A valid name consists of one or more letters and can contain digits and underscores. Each Waveshaper object must have a unique name
wsconfig	[in]	Path (to Configuration files)

Returns:

Result code, WS_SUCCESS if success, otherwise it is an error code.

Result Code	Description
WS_SUCCESS	Command successfully executed
WS_DUPLICATE_NAME	WaveShaper name already in uset

Example : Create WaveShaper instance without call back function

```
/* Create WaveShaper instance without call back function */
int rc = ws_create_waveshaper_forsimulation("ws1", "sn007090.wsconfig");
if (rc == WS_SUCCESS)
; WaveShaper object created
else
; Error handling code
```

8.5 PS Command Overview and Program Structure

Before using any WaveShaper PowerSplitting API commands, the application must first create a PowerSplitting instance and, at the application termination sequence, it must close and delete the PowerSplitting instance. The process overview is shown in flow chart below.

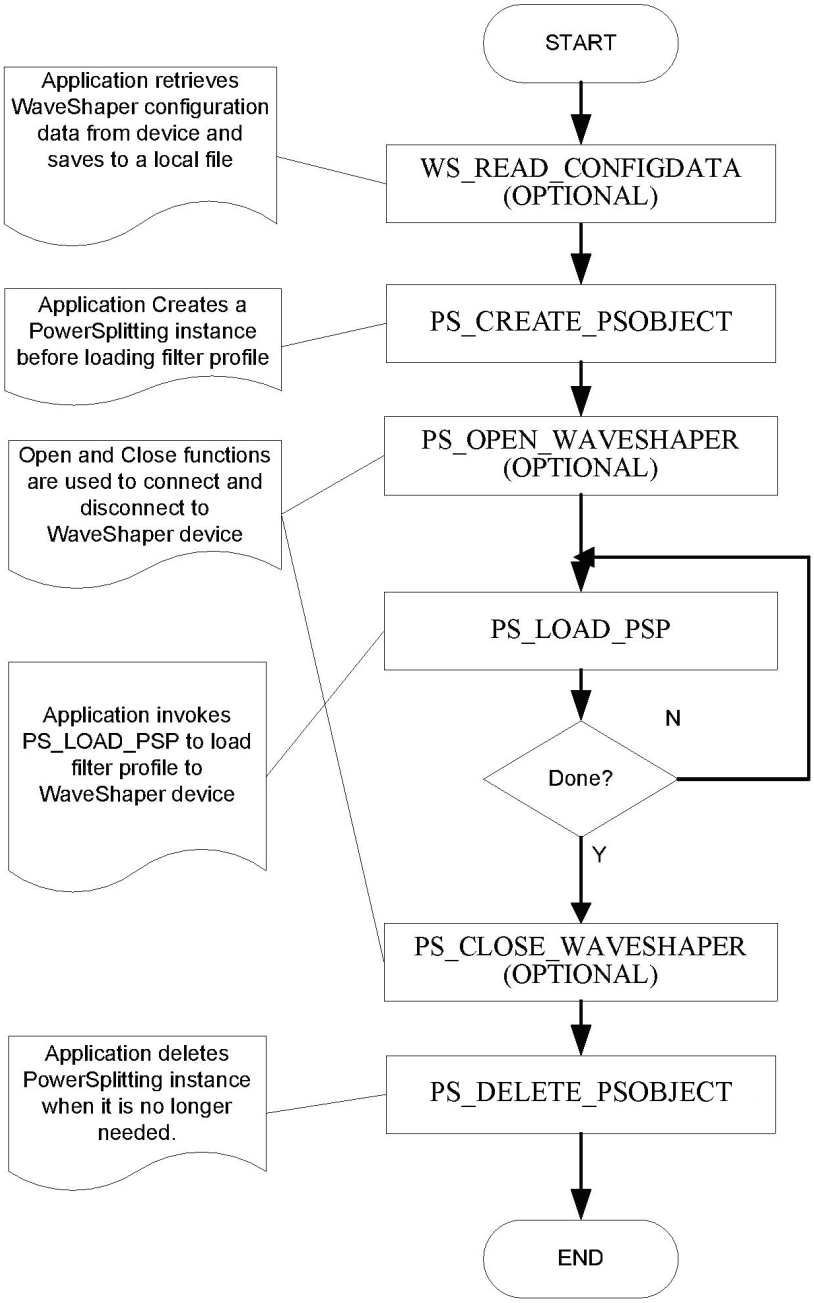


Figure 8.4 Program Structure Flow Chart

8.5.1 C Sample Code : POWER SPLITTING LOAD PSP

```
/** @file LoadPSP.cpp
* @author Finisar Australia - Copyright (c) 2005-2011
*
* This is a sample program which shows how to load a PSP filter profile
* through the ps_load_psp command.
* It also demonstrates how to initialize the API and create the
* PowerSplitting instance.
*/
#include "ws_api.h"
#include "ws_psapi.h"
#include <stdio.h>
#include <string>
void CheckError(int rc);
int main(int argc, const char* argv[])
{
int rc;
if ( argc != 3 ) /* argc should be 2 for correct execution */
{
/* We print argv[0] assuming it is the program name */
printf( "usage: %s wsconfig pspfile", argv[0] );
return 0;
}
//Load PSP file:
FILE* fp = fopen( argv[2], "r");
if(fp == NULL) {
printf("Error: cannot open file %s\n", argv[2]);
return -1;
}
// obtain file size:
fseek(fp, 0 , SEEK_END);
long lSize = ftell(fp);
rewind(fp);
char* profiletext = new char[lSize];
rc = fread(profiletext, 1, lSize, fp);
profiletext[rc] = '\0';
rc = ps_create_psobject("PS1", argv[1]);
CheckError(rc);
if(rc == WS_SUCCESS) {
rc = ps_load_psp("PS1", profiletext);
CheckError(rc);
}
if(rc == WS_SUCCESS) {
rc = ps_delete_psobject("PS1");
CheckError(rc);
}
delete[] profiletext;
return 0;
}

void CheckError(int rc) {
if(rc!=0)
printf("Error Detected! (rc=%d)\nDescription: %s\n", rc,
ws_get_result_description(rc));
}
```

8.5.2 Python Sample Code: POWER SPLITTING LOAD PSP

```
#import wsapi python wrapper
from wsapi import *
#create waveshaper instance and name it "PS1"
rc = ps_create_psobject("PS1", "SN025399.wsconfig")
print "ps_create_psobject rc="+ws_get_result_description(rc)
#read profile from WSP file
PSPfile = open('MyProfile.psp', 'r')
profiletext = PSPfile.read()
#compute filter profile from profile text, then load to Waveshaper device
rc = ps_load_psp("PS1", profiletext)
print "ps_load_psp rc="+ws_get_result_description(rc)
#delete the waveshaper instance
rc = ps_delete_psobject("PS1")
print "ps_delete_psobject rc="+ws_get_result_description(rc)
```

8.6 PS API Functions

8.6.1 PS_CREATE_PSOBJECT

int ps_create_psobject (const char * name, const char * wsconfig)

Creates a WaveShaper PowerSplitting object instance with a user specified name. The user can choose any name containing one or more letters or digits or underscore, provided that distinct names are used for each WaveShaper device.

Note that PS_CREATE_PSOBJECT object will not open the communication port or make connections to the WaveShaper device. Opening a connection to the device is done by invoking the PS_OPEN_WAVESHAPER function.

Parameters:

name	[in]	User specified PowerSplitting name A valid name consists of one or more letters and can contain digits and underscores. Each PowerSplitting object must have a unique name
wsconfig	[in]	Path (to Configuration files)

Results:

Result code, WS_SUCCESS if success, otherwise it is an error code.

Result Code	Description
WS_SUCCESS	Command successfully executed
WS_DUPLICATE_NAME	PowerSplitting name already in use
WS_NOT_SUPPORTED	The WaveShaper is not compatible with Power Splitting.

Example : Create PowerSplitting instance without call back function

```
int rc = ps_create_psobject("PS1", "sn007090.wsconfig");
if (rc == WS_SUCCESS)
; PowerSplitting object created
else
; Error handling code
```

8.6.2 PS_DELETE_PSOBJECT

int ps_delete_psobject (const char * name)

This command deletes the PowerSplitting object. The PowerSplitting object will be automatically closed, if it is in open state.

Parameters:

name	[in]	Previously created PowerSplitteing name

Results:

Result code, WS_SUCCESS if success, otherwise it is an error code.

Result Code	Description
WS_SUCCESS	Command successfully executed
WS_WAVESHAPER_NOT_FOUND	PowerSplitting object does not exist

Example :

```
int rc;
rc = ps_create_psobject("PS1","sn007090.wsconfig");
...
rc = ps_delete_psobject("PS1");
```

8.6.3 PS_OPEN_WAVESHAPER

int ps_open_waveshaper (const char * name)

This command opens an existing PowerSplitting object and establishes the connection to the defined WaveShaper.

Note that it is optional to call this function before invoking profile downloading functions. WaveShaper API will automatically open the connection before downloading profile if it is not opened yet.

However, it is recommended to invoke “ps_open_waveshaper” explicitly for better performance, when downloading multiple filter profiles. Downloading speed will be increased, as no re-connection is needed between downloading each new profile.

Parameters:

name	[in]	Previously created PowerSplitting name

Results:

Result code, WS_SUCCESS if success, otherwise it is an error code.

Result Code	Description
WS_SUCCESS	Command successfully executed
WS_WAVESHAPER_NOT_FOUND	PowerSplitting object does not exist
WS_OPENFAILED	Could not open the WaveShaper. Possible connection problem.

Example :

```
int rc = ps_open_waveshaper("PS1");
if (rc == WS_SUCCESS)
      ; Connection to WaveShaper successfully established
else
      ; Error handling code
```

8.6.4 PS_CLOSE_WAVESHAPER

int ps_close_waveshaper (const char * name)

Close WaveShaper. Disconnect from the WaveShaper device.

Note that it is optional to call this function. The connection will be automatically

closed during deletion of PowerSplitting object (see PS_DELETE_PSOBJECT).

Parameters:

name	[in]	Previously created PowerSplitting name

Results:

Result code, WS_SUCCESS if success, otherwise it is an error code.

Result Code	Description
WS_SUCCESS	Command successfully executed
WS_WAVESHAPER_NOT_FOUND	PowerSplitting object does not exist

Example :

```
int rc = ps_close_waveshaper("PS1");
if (rc == WS_SUCCESS)
      ; Connection to WaveShaper successfully closed
else
      ; Error handling code
```

8.6.5 PS_LOAD_PSP

int ps_load_psp (const char * name, const char * profiletext)

Apply PSP filter and wait for completion. Calculate the filter profile based on PSP-text, then load filter profile to WaveShaper device.

Parameters:

name	[in]	Previously created PowerSplitting name
profiletext	[in]	PSP text string

Results:

Result code, WS_SUCCESS if success, otherwise it is an error code.

Result Code	Description
WS_SUCCESS	Command successfully executed
WS_WAVESHAPER_NOT_FOUND	PowerSplitting object does not exise
WS_INVALIDPORT	Port number is not valid
WS_INVALIDFREQ	Frequency specified out of range
WS_INVALIDATTN	Attenuation is not valid (e.g. negative value)
WS_INVALIDSPACING	Frequencies not incremented in 0.001 THz step
WS_NARROWBANDWIDTH	Bandwidth of frequencies to the same port is less than 0.010 THz
WS_INVALIDPROFILE	Other parsing error
WS_OPENFAILED	Could not open the WaveShaper. Possible connection problem.
WS_WAVESHAPER_CMD_ERROR	Error response from WaveShaper. May be communication corruption.

Example :

```
/* hard coded profile text */
char* profiletext=
"#1,2\n
#Unscaled\n
192.000\t1.0\t0.0\t2.0\t3.14\n
192.001\t1.0\t0.0\t3.0\t3.14\n...";
int rc = ps_load_psp("PS1", profiletext);
if (rc == WS_SUCCESS)
; Profile loaded
else
; Error handling code
```

8.6.6 PS_LOAD_PREDEFINEDPROFILE

```
int ps_load_predefinedprofile(const char* name, int filtertype, float center, float bandwidth, float attn, int port)
```

This function behaves in the exact same way as ws_load_predefinedprofile (See 4.6). It must be called instead of ws_load_predefinedprofile when using instances of the PowerSplitting object.

8.6.7 PS_GET_FREQUENCYRANGE

```
int ps_get_frequencyrange(const char* name,float* start, float* stop)
```

Get start and stop frequency of the WaveShaper. This allows the user to determine, for instance, if the attached WaveShaper is for C- or L-band operation. This is also a useful check to ensure that a PSP does not extend beyond the operating range of the unit being controlled.

Parameters:

name	[in]	Previously created WaveShaper name
start	[out]	Starting Frequency (THz)
stop	[out]	Stopping Frequency (THz)

Returns:

Result code, WS_SUCCESS if success, otherwise it is an error code.

Result Code	Description
WS_SUCCESS	Command successfully executed
WS_WAVESHAPER_NOT_FOUND	PowerSplitting object does not exist

Example :

```
float start, stop;
int rc = ps_get_frequencyrange ("PS1", &start, &stop);
printf("Frequency range: %f-%f\n", start, stop);
```

8.7 Error Code Type Definitions

#define WS_SUCCESS (0) //success

#define WS_ERROR (-1) //general Error

#define WS_INTERFACE_NOTSUPPORTED (-2) //interface not supported

#define WS_NULL_PARAM (-3) //null input

#define WS_UNKNOWN_NAME (-4) //name cannot be resolved

#define WS_NO_ITEM (-5) //no such item

#define WS_INVALID_CID (-6) //invalid class id

#define WS_INVALID_IID (-7) //invalid interface id

#define WS_NULL_POINTER (-8) //null pointer

#define WS_BUFFEROVERFLOW (-9) //buffer overflow

#define WS_WRONGSTATE (-10) //wrong state

#define WS_NO_THREADPOOL (-11) //no thread pool

#define WS_NO_DIRECTORY (-12) //no directory

#define WS_BUSY (-16) //busy

#define WS_NULL_BUFFER (-17) //buffer is null

#define WS_NO_SUCH_FIELD (-18) //no such field

#define WS_NO_SUCH_PROPERTY (-19) //no such property

#define WS_IO_ERROR (-20) //IO error

#define WS_TIMEOUT (-21) //timeout

#define WS_ABORTED (-22) //aborted

#define WS_LOADMODULE_ERROR (-23) //load module failed

#define WS_GETPROCESS_ERROR (-24) //get process error

#define WS_OPEN_PORT_FAILED (-25) //failed to open port

#define WS_NOT_FOUND (-26) //not found

#define WS_OPEN_FILE_FAILED (-27) //failed to open file

#define WS_FILE_TOOLARGE (-28) //file size too large

#define WS_INVALIDPORT (-29) //invalid port number

#define WS_INVALIDFREQ (-30) //invalid frequency

#define WS_INVALIDATTN (-31) //invalid attenuation

#define WS_INVALIDPROFILE (-32) //other profile error

#define WS_INVALIDSPACING (-33) //invalid freq space

```
#define WS_NARROWBANDWIDTH (-34) //bandwidth < 0.010 THz

#define WS_OPENFAILED (-35) //open failed

#define WS_OPTION_ERROR (-36) //option error

#define WS_COMPRESS_ERROR (-37) //compress error

#define WS_WAVESHAPER_NOT_FOUND (-38) //WaveShaper not found

#define WS_WAVESHAPER_CMD_ERROR (-39) //command to ws error

#define WS_NOT_SUPPORTED (-40) //function not supported

#define WS_DUPLICATE_NAME (-41) //duplicate name

#define WS_INVALIDFIRMWARE (-42) //invalid firmware format

#define WS_INCOMPATIBLEFIRMWARE (-43) //firmware ver incompatible

#define WS_OLDERFIRMWARE (-44) //firmware ver too old
```

9.1 WaveShaper M Series General Specifications

9.2 WaveShaper S Series General Specifications

9.3 Optical Specifications

Section 9: WaveShaper Specifications

9.1 WaveShaper M Series General Specifications

These specifications apply to the model/part numbers listed in Table 9.1.

Model	Part Number	Product Code	Band
100M	1139584	WS-AA-0100M-ZZ-H	C-Band
100M/L	1169553	WS-AA-0100M-LB-H	L-Band
120M	1132579	WS-AA-0120M-ZZ-H	C-Band
1000M	1116811	WS-AA-1000M-ZZ-H	C-Band
	1140071	WS-AA-1000M-ZZ-F	
1000M/L	1123168	WS-AA-1000M-LB-H	L-Band
	1140711	WS-AA-1000M-LB-F	
1000M/X	1139588	WS-AA-1000M-XB-H	C+L-Band
1000M/SP	1139590	WS-AA-1000M-SP-H	C-Band
4000M	1123167	WS-AA-4000M-ZZ-H	C-Band
	1142399	WS-AA-4000M-ZZ-F	
4000M/L	1123169	WS-AA-4000M-LB-H	L-Band
4000M/X	1140997	WS-AA-4000M-XB-H	C+L-Band

Table 9.1 Specification Part Number List

9.1.1 Absolute Maximum Ratings

Exceeding the ranges defined here may result in permanent damage to the module

Parameter	Units	Min	Max	Notes
StorageTemperature	ºC	-20	+85	
Humidity	%	5	95	Non-Condensing
Cooling Air Temperature	ºC		35	
Power Consumption	W		50	
ESD	kV		16	See note(1)

Table 9.2 Absolute Maximum Ratings

Note 1 : Class N per HBM, meets IEC 61000-4-2 level 4.

9.1.2 Normal Operating Conditions

Parameter	Units	Min	Max	Notes
Case Temperature	ºC	15	55	See Note 1
Humidity	%	10	90	Non-Condensing
Total Optical Input Power	dBm		27	
Per-Channel Optical Input Power	dBm		13	For any 50 GHz channel.
Cooling Airflow Rate	m/s	1		Along direction of fins
Cooling Air Temperature	ºC		35	
Warm up Time	mins		10	At $T_{ambient}$ = 25 ºC

Table 9.3 Normal Operating Conditions

Note 1 : Measured at centre of module case heat sink.

9.1.3 Electrical Specifications

Parameter	Units	Min	Typ	Max	Notes
Power Consumption	W		19	50	After Warm-up period
Power Supply	V	5	5.2	5.6	See note (1).
Electrical Connector	Phoenix 1803277				Mates with Phoenix 1803578
Supported Interfaces	Mini-USB 2.0				

Table 9.4 Electrical and Communication Specifications

Note 1: Ensure the that the supplied voltage at the point of the connector is between 5 V and 5.6 V.

The wire used should be red for +5 V and and black for 0 V 22 AWG (0.5mm,16/0.2,Tri-rated PVC insulated UL1015).

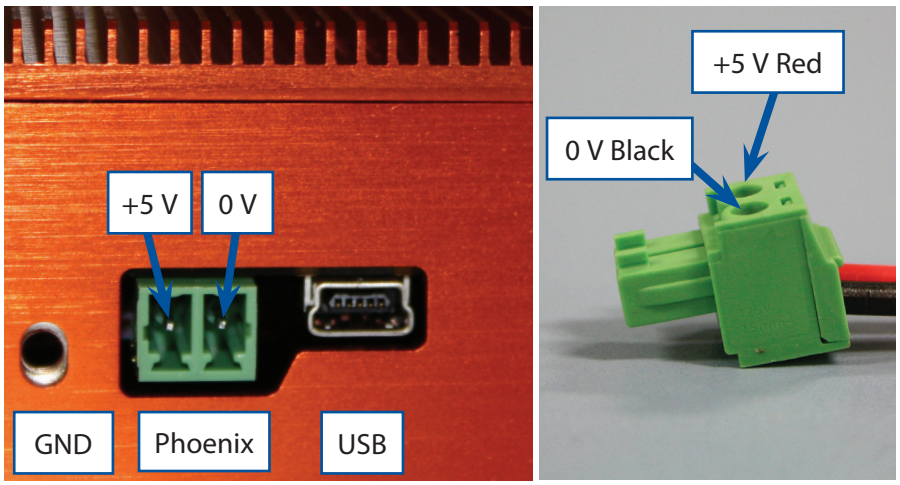
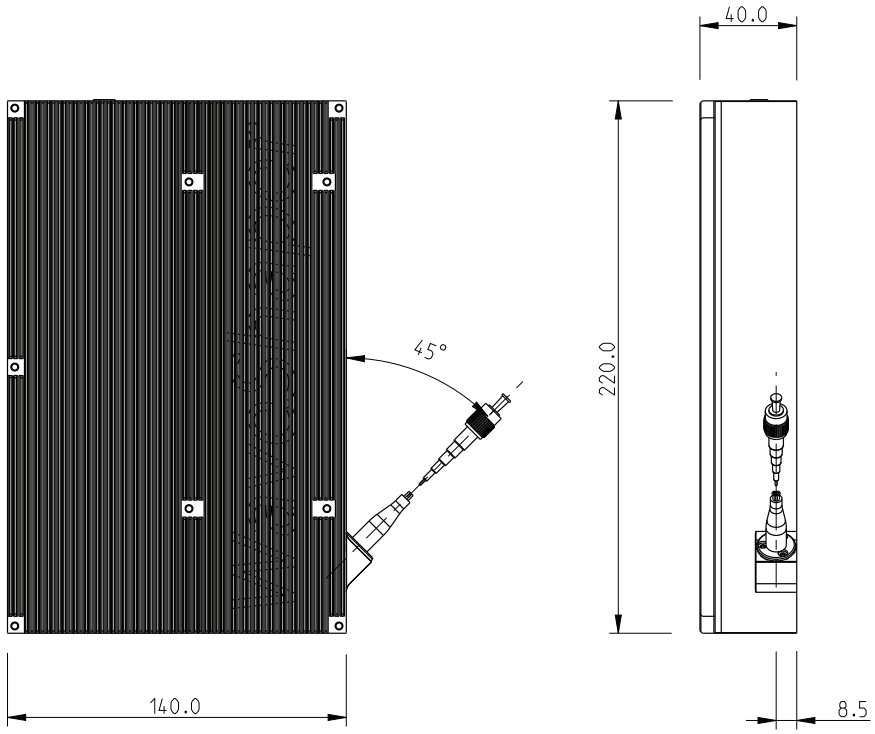


Figure 9.1 Phoenix Connector Power Wiring

9.1.4 Mechanical Specifications



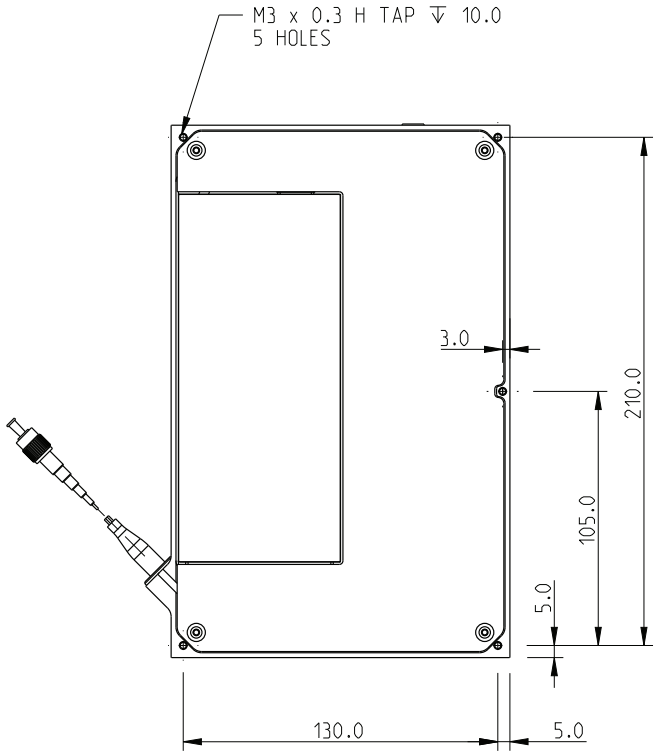


Figure 9.2 Technical drawings of the M-series WaveShaper models.

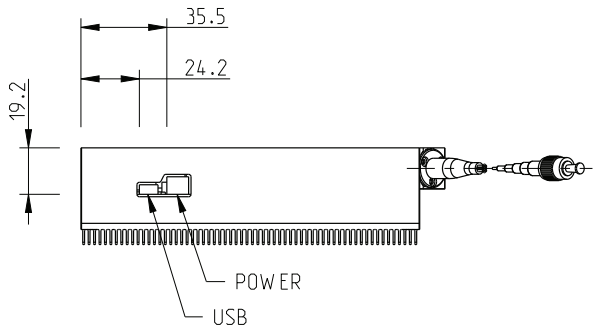


Figure 9.3 Technical drawing of the M-series WaveShaper models, side view.

To ensure correct and safe operation of the WaveShaper, the WaveShaper module should be positioned such that:

- The Module is placed with sufficient free space above the heatsink surface. to ensure correct airflow across the unit for cooling purposes. See section 9.1.2.
- There is clear access to the input and output fibres to ensure that the fibre connectors can be safely connected and disconnected as required.

9.2 WaveShaper S Series General Specifications

These specifications apply to the model/part numbers listed in Table 9.5.

Model	Part Number	Product Code	Band
100S	1126428	WS-AA-0100S-ZZ-H	C-Band
	1148770	WS-AA-0100S-RM-H	
100S/L	1169552	WS-AA-0100S-LB-H	L-Band
1000S	1097198	WS-AA-1000S-ZZ-H	C-Band
	1134295	WS-AA-1000S-ZZ-F	
	1111606	WS-AA-1000S-RM-H	
1000S/L	1102268	WS-AA-1000S-LB-H	L-Band
	1140712	WS-AA-1000S-LB-F	
1000S/X	1139587	WS-AA-1000S-ZZ-H	C+L-Band
1000S/SP	1139589	WS-AA-1000S-SP-H	C-Band
4000S	1102302	WS-AA-4000S-ZZ-H	C-Band
	1142398	WS-AA-4000S-ZZ-F	
	1111605	WS-AA-4000S-RM-H	
4000S/L	1102303	WS-AA-4000S-LB-H	L-Band
4000S/X	1140996	WS-AA-4000S-XB-H	C+L-Band
16000S	1214298	WS-AA-16000S-ZZ-D	C-Band
	1202135	WS-AA-16000S-ZZ-D	

Table 9.5 Specification Part Number List

9.2.1 Absolute Maximum Ratings

Exceeding the ranges defined here may result in permanent damage to the module.

Parameter	Units	Min	Max	Notes
StorageTemperature	ºC	-20	+50	
Humidity	%	5	95	Non-Condensing
Cooling Air Temperature	ºC		35	
Power Consumption	VA		40	
ESD	kV		16	Class N per HBM, meets IEC 61000-4-2 level 4

Table 9.6 Absolute Maximum Ratings

9.2.2 Normal Operating Conditions

Parameter	Units	Min	Max	Notes
Operating Temperature	ºC	15	35	See note(1)
Humidity	%	10	90	Non-Condensing
Total Optical Input Power	dBm		27	
Per-Channel Optical Input Power	dBm		13	For any 50 GHz channel.
Cooling Air Temperature	ºC		35	See note(1)
Warm up Time	mins		10	At $T_{ambient}$ = 25 ºC

Table 9.7 Normal Operating Conditions

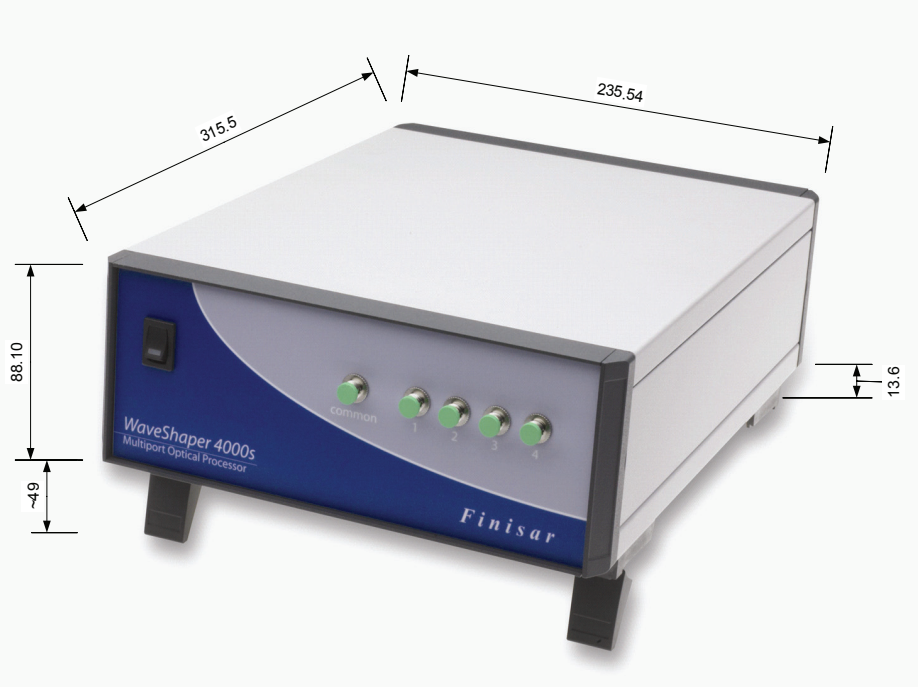
Note 1: Air temperature at input to vents on base of unit

9.2.3 Electrical Specifications

Parameter	Units	Min	Typ	Max	Notes
Power Consumption	VA		20	50	At $T_{ambient}$ = 25 ºC
Power Supply Voltage	V	100		240	
Power Supply Frequency	Hz	50		60	
Electrical Connector	IEC				Suitable Power cord included
Supported Interfaces	USB 2.0				USB Cable A-male to B-male supplied

Table 9.8 Electrical and Communication Specifications

9.2.4 Mechanical Specifications



To ensure correct and safe operation of the WaveShaper, the WaveShaper chassis should be positioned such that:

- The rear of the chassis is at least 50 mm from any obstruction. This is required to ensure correct airflow through the unit for cooling purposes.
- There is clear access to the mains power socket to ensure that the mains cable can be safely connected and disconnected as required.
- There is clear access to the input and output fibres on the front panel to ensure that the fibre connectors can be safely connected and disconnected as required.

	120M, 120S	100M, 100S	100M/L, 100S/L	1000S, 1000M 4000S, 4000M 16000S	1000S/L, 1000M/L 4000S/L, 4000M/L	1000S/X, 1000M/X 4000S/X, 4000M/X	1000S/SP, 1000M/SP
Operating Frequency Range	191.250 - 196.275 THz (1527.4 - 1567.5nm)	191.250 - 196.275 THz (1527.4 - 1567.5nm)	186.350 - 191.000 THz (1569.6 - 1608.7 nm	191.250 - 196.275 THz (1527.4 - 1567.5nm)	186.350 - 191.000 THz (1569.6 - 1608.7 nm)	187.275 -196.275 THz (1527.4 - 1600.8 nm)	191.250 - 196.275 THz (1527.4 - 1567.5nm)
Filter Bandwidth	5 THz	25 GHz -200 GHz	25 GHz -200 GHz	10 GHz - 5 THz (0.08 - 40 nm)	10 GHz - 4.65 THz (0.08 nm - 35 nm)	20 GHz - 9 THz	10 GHz - 5 THz
Filter Shape	Arbitrary	Band-pass, Gaussian		Arbitrary			
Frequency Setting Resolution	±1 GHz (±8 pm)						
Frequency Setting Accuracy	±2.5 GHz (±20 pm)					±5 GHz	±2.5 GHz
Bandwidth Setting Resolution	n/a	±1 GHz (±8 pm)					
Bandwidth Setting Accuracy	n/a	± 5 GHz (±40 pm)				±10 GHz	± 5 GHz
Bandwidth Setting Repeatability	n/a	± 2.5 GHz (±20 pm)				± 5 GHz	±2.5 GHz
Group Delay Control Range	n/a	n/a		-25 ps to +25 ps		-15 ps to +15 ps	-25 ps to +25 ps
Attenuation Control Range	10 dB	n/a		0 - 35 dB			
Attenuation Setting Resolution	0.01 dB	n/a		0.01 dB			
Attenuation Setting Accuracy	±1.0 dB	n/a		±1.0 dB from 0-10 dB, ±10 % from 10-30 dB			
Insertion Loss (typ)	4.5 dB						4.5dB See note (1)
Insertion Loss Non-Uniformity (typ)	0.5 dB					1 dB	0.5 dB
PDL (typ)	0.2 dB						n/a
Return Loss (typ)	>25 dB						
Max Total Optical Input Power	+ 27 dBm						
Max Optical Power per 50GHz Channel	+13 dBm						

Note 1: Measured on signal in slow axis.