WaveShaper API Command Set
Reference Manual
Release 2.5

Specifications documented here are subject to change.
Please contact your account manager to obtain the latest revision.

REVISION HISTORY

PRE RELEASE REV		DESCRIPTION OF CHANGE	ORIGINATOR
A		• Initial Issue	Qing Li
B		• Reviewed and adapted for WSP	Michael Roelens
C		• Added version information to API	Qing Li
D		• Added firmware upload function to API	Qing Li
E		• Added more functions to API	Qing Li
F		• Added modeling functions to API	Qing Li
G		• Minor change to flow chart	Qing Li
H		• Added more functions to API	Qing Li
I		• Added flow of retrieve WSCONFIG	Qing Li
J		• Added Power Splitting API	Patrick Blown
K		• Added ws_create_waveshaper4	Qing Li
REV	**ECO#**	**DESCRIPTION OF CHANGE**	**ORIGINATOR**
A00	138564	• Initial formal release	Joseph Zagari

Table 1: Revision History

TABLE OF CONTENTS

1 INTRODUCTION

This document describes the WaveShaper API functions which 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. The API is available on Linux and Windows operating systems.

This manual contains an overview of the WSP file format used by the API (Chapter 2), followed by an outline of how to structure a program to control a WaveShaper using the API (Chapter 0). Chapter 4 provides a complete list of all the commands, together with sample C code for each function. The Appendices provide type definitions of the API (Appendix A) together with a File List (appendix B) and additional programming examples in C and Python (Appendix C).

2 THE WSP FORMAT

The WaveShaper Preset (WSP) file format allows the user to specify the spectral amplitude and phase of the WaveShaper optical response, and, in the case of the WaveShaper 4000S, direct the output light to a particular port. The format of a WSP filter is a tab delimited text string with four columns: Absolute Frequency (THz), Attenuation (dB), Phase (rad) and Port Number. The following rules must be followed to create a valid WSP filter/switch specification:

1. The frequencies are defined in absolute values, in units of THz, and with a resolution of 0.001 THz (1 GHz). The frequency range which can be specified must be less than, or equal to, the operating range of the WaveShaper. (For example, for C-band WaveShapers, the frequency range in the file should be within the range of 191.250 to 196.275 THz.) Frequency values must increment in 0.001 THz (1 GHz) steps. A partial definition that covers a continuous range within the valid frequency range is also allowed.
2. The port needs to be defined in the fourth column. Selecting Port 0 sets that frequency to "Block". Please ensure that the ports specified in this column are ports that are available on the WaveShaper ("0" and "1" are valid in the case of a WaveShaper 1000, values 0-4 are valid for a WaveShaper 4000).
3. The minimum bandwidth of each band of frequencies that is to be sent to a particular output port needs to be at least 0.010 THz (10 GHz).
4. The WSP input file will be checked during ws_load_profile function call. An error code will be returned if the WSP input fails during the validation process. This is discussed in detail in Section 4.5

An example of a WSP that covers the frequency range from 191.250 THz to 191.260 THz is shown below.

Frequency	Attenuation	Phase	Port (NOTE: Header is not part of WSP text)
191.250	22.8	1.5707	1
191.251	21.4	1.5707	1
191.252	20.4	1.5707	1
191.253	19.6	1.5707	1
191.254	19.0	1.5707	1
191.255	18.5	1.5707	1
191.256	18.1	1.5707	1
191.257	17.7	1.5707	1
191.258	17.4	1.5707	1
191.259	17.2	1.5707	1

The valid range of arguments values for a WSP file is shown in the table below:

Parameter	**Minimum**	**Maximum**	**Notes**
Frequency	WaveShaper Minimum Frequency	WaveShaper Maximum Frequency	These can be interrogated from the unit using the ws_get_frequencyrange command (See Section 4.9)
Attenuation	0	40	1. The requested Attenuation must be a positive number in the range 0 to 40 dB. 2. Requested Attenuation values of greater than 40 dB will be set to the 'Block' state (nominally 50 dB). 3. Requested Attenuations between 35 and 40dB will be attempted to be set but if the requested attenuation is greater than the capability of the WaveShaper chassis, the attenuation will be set to the 'Block' state for that frequency. 4. Attempts to program gain (requested attenuation less than 0 dB) will be truncated to 0 dB.
Phase	0	2π	The WSP may specify a phase outside of this range, however, this will be re-calculated by the WaveShaper software on interpolation as (phase modulo 2π).
Port	0	1 (WS 1000) 4 (WS 4000)	Port 0 = block

The following table summarizes the valid, calibrated ranges for the parameters on the WaveShaper 120m. Values outside the specified range will result in an error:

Parameter	Minimum	Maximum	Notes
Frequency	WaveShaper Minimum Frequency	WaveShaper Maximum Frequency	These can be interrogated from the unit using the ws_get_frequencyrange command (See Section 4.9)
Attenuation	0	10	Attempts to program attenuations out of the range will result in an error
Phase	0	0	Attempts to program phase out of the range will result in an error
Port	1	1	Attempts to program port out of the range will result in an error

Furthermore, it is possible to set the WaveShaper 120m to a complete block state, by calling the WS_LOAD_PREDEFINEDPROFILE function (see section 4.6).

DEVELOPMENT ENVIRONMENT SETUP

User can develop customized WaveShaper applications with different programming languages. C and Python API are provided. If other programming language is used, user will need to wrap the C API in that language. The following sections describes how to setup the development environment in C and python.

2.1 WAVESHAPER C API

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

File Name	Description
waveshaper\api\include\ws_api.h	Header file contains all function and type definitions.
waveshaper\api\wsapi.dll (.so)	WaveShaper API dynamic link library.
waveshaper\api\wsapi.lib	Static import library to wsapi.dll (Windows only)

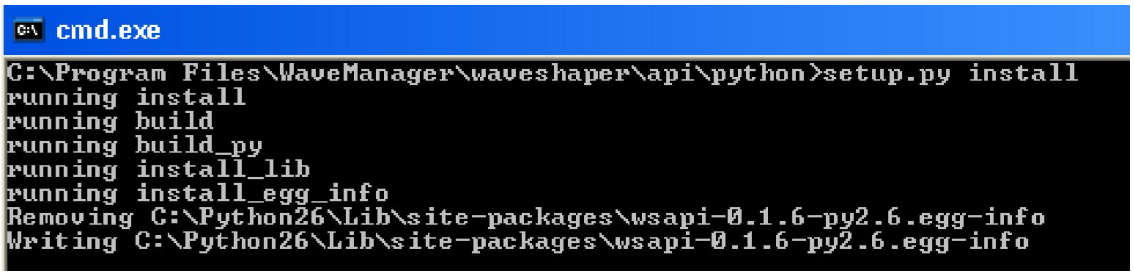
The header file should be included in the C code that uses WaveShaper API. The static import library is used by the linker to resolve WaveShaper API symbols during linking process.

The following table lists the dependencies of wsapi.dll. All of them must be put into 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

2.2 WAVESHAPER PYTHON API

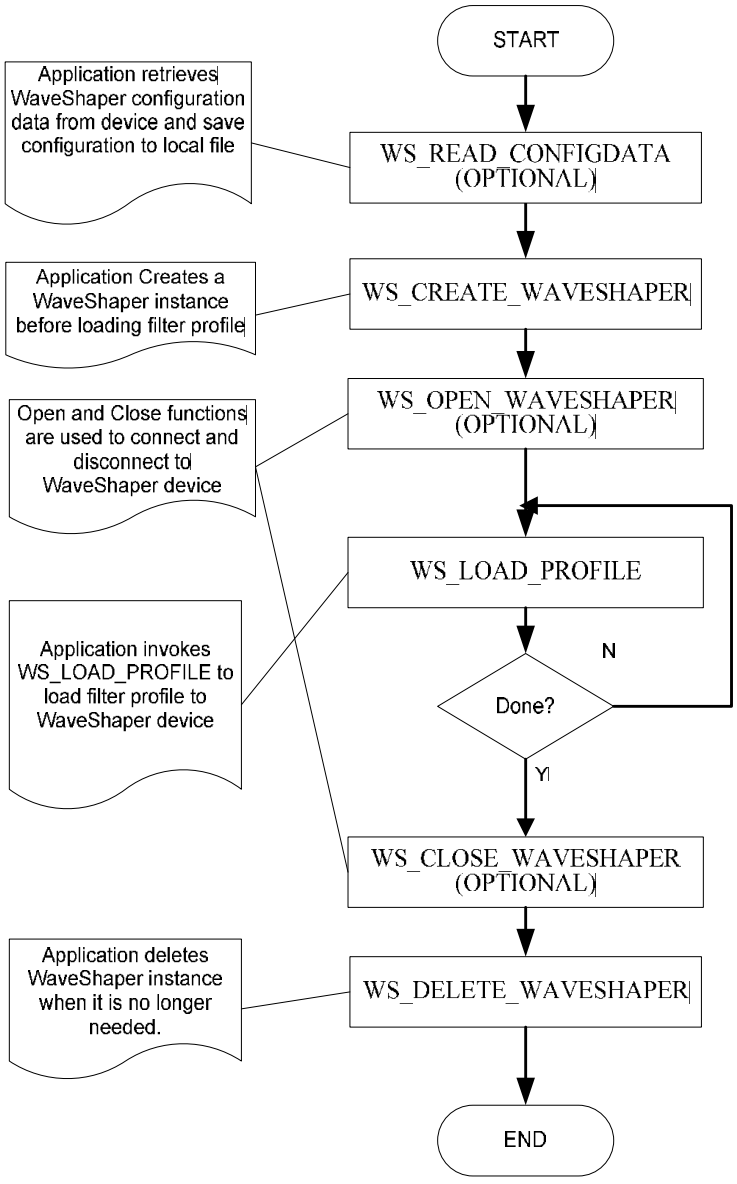
A Python API is provided to support accessing WaveShaper functions from Python scripts. The module setup script is located in waveshaper\api\python sub-directory. The "*setup.py*" script is provided to install WaveShaper API python module into user's python environment. An example of invoking the setup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



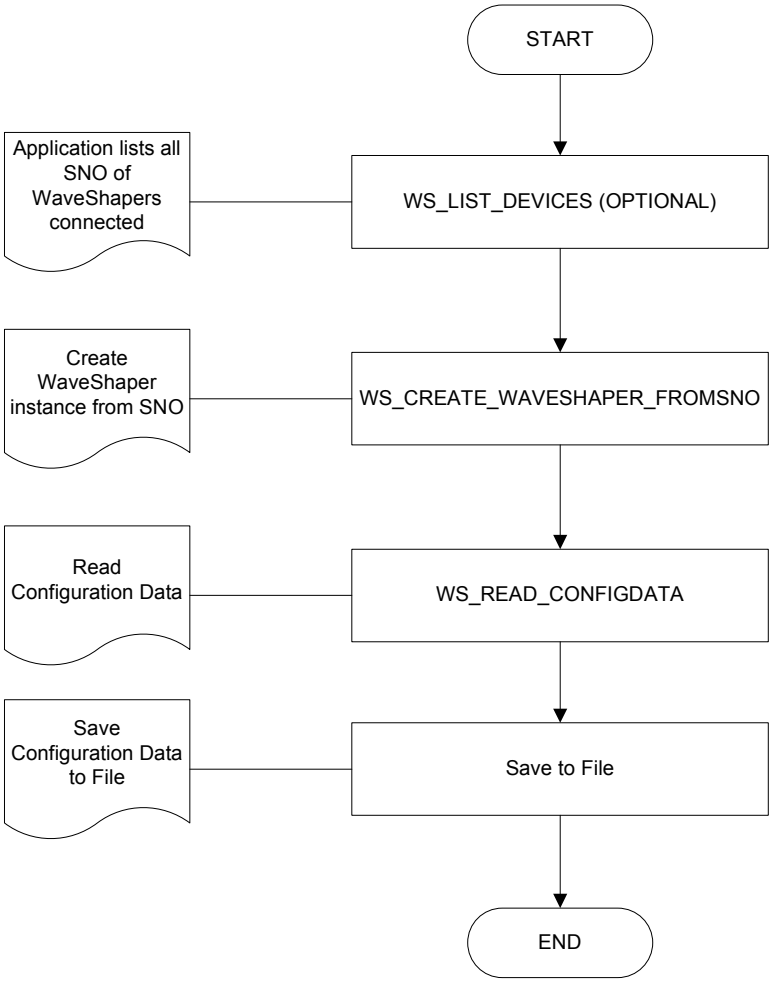
The file *wsapi.dll* and all DLL dependencies (*FTD2XX.DLL(so)* and *ws_cheetah.DLL*) must be put into directory that is included in the system's dynamic library loading path, so that the user's python script can access WaveShaper API functions.

3 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.



WaveShapers with updated firmware support configuration data embedding. User can download WaveShaper configuration data from the device by following the sequence below.



4 API FUNCTION DOCUMENTATION

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 WS_OPEN_WAVESHAPER function.

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.
wsconfig [in]	Path string to Configuration file.

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 use

Example 1: 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
```

4.1 WS_CREATE_WAVESHAPER4

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

Creates a WaveShaper object instance with a user specified name and sub-configuration part. It is similar to WS_CREATE_WAVESHAPER. The cfg parameter selects the sub-configuration part on WaveShaper with multiple configuration regions.

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.
wsconfig [in]	Path string to Configuration file.
cfg [in]	Select sub-configuration part

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 use

Example 1: 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
```

4.2 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

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

4.3 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] Previously created WaveShaper name

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

4.4 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

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

4.5 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

Returns:

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

Error 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
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. May be a 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
```

4.6 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
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, WS_SUCCESS if success, otherwise it is an error code.

Error 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. May be a connection problem.
WS_WAVESHAPER_CMD_ERROR	Error response from WaveShaper. May be 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
```

4.7 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

Example:

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

4.8 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 [out]       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 doesn't exist

Example:

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

4.9 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 [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	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);
```

4.10 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 or WaveShaper 4000S

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);
```

4.11 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 [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_profile("ws1", buffer, &size);
printf("profile: %s\n", buffer);
free(buffer);
```

4.12 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);
```

4.13 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);
```

4.14 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 new 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
```

4.15 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
```

4.16 WS_CREATE_WAVESHAPER_FROMSNO

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

This function creates a named WaveShaper object instance from a device serial number. The user can choose any name containing one or more letters or digits or underscores, 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 the device.

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

Parameters:

name	[in]	User specified WaveShaper name. A valid name contains one or more letters or digits or underscores. 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));
```

4.17 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 byte 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
```

4.18 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 to hold embedded data read from device
size	[in]	Buffer Size
nwrite	[out]	Number of byte written

Returns:

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

Example:

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

4.19 WS_LOAD_PROFILE_FOR_MODELING

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

Load a WSP filter to the internal buffer for modeling and 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.

Error 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
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
```

4.20 WS_GET_MODEL_PROFILE

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

Get the WSP representation of the 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 `psize` variable will be updated with the actual WSP text length. If the output size is larger than the 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);
```

4.21 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

```
char response[256];
int  responsesize = sizeof(response);
memset(response, 0, sizeof(response));
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
```

5 POWER SPLITTING API

This API provides a set of commands allowing the WaveShaper to produce power splitting profiles. This enables the WaveShaper 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.

All commands prefixed with ps form part of the Powers Splitting API. They are a separate set of API commands and cannot be used in conjunction with any commands prefixed with 'ws', with the exception of 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.

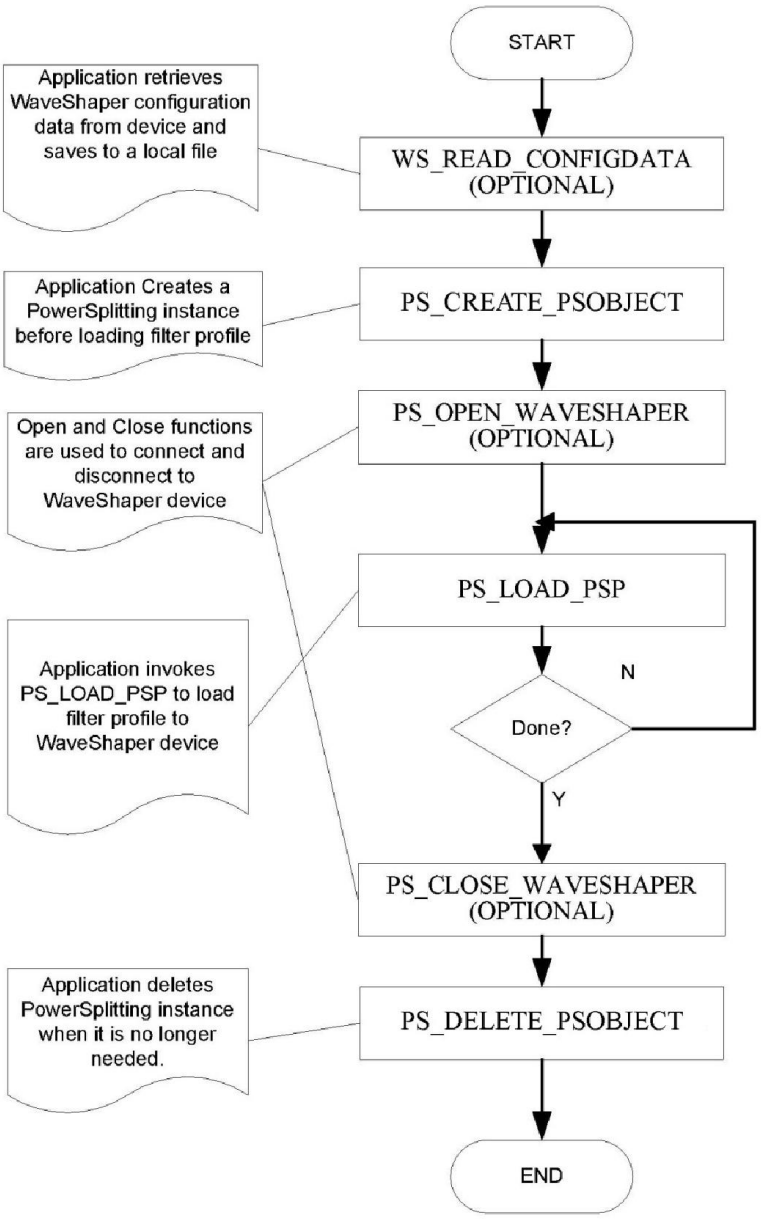
5.1 SETUP

SETUP FOR C
Include the additional header file ws_psapi.h at the top of your C code.

SETUP FOR PYTHON
The set of Power Splitting API commands are provided as part of the standard WaveShaper API Python installation. See Section 2.2 for more information about setting up Python.

5.2 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.



6 THE PSP FORMAT

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. The file can be generated using a spreadsheet program, such as Microsoft Excel (saving the document as a tab delimited text file).

To ensure data integrity, the PowerSplitting API 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.

6.1 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 Section 6.2 for more information).

	Port 1		Port 2		Port 3		Port 4	
Frequency [THz]	Attenuation [dB]	Phase [Rads]	Attenuation [dB]	Phase [Rads]	Attenuation [dB]	Phase [Rads]	Attenuation [dB]	Phase [Rads]
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

* *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 PowerSplitting API 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 PowerSplitting API.

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 PowerSplitting API on interpolation as (phase modulo 2π).

6.2 (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 attenuation and phase column pair in the body. I.e. the nth 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 POWERSPLITTING FUNCTION DOCUMENTATION

7.1 PS_CREATE_PSOBJECT

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

Creates a 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 PS_OPEN_WAVESHAPER function.

Parameters:

name [in]	User specified PowerSplitting name. A valid name contains one or more letters or digits or underscore. Each PowerSplitting object must have a unique name.
wsconfig [in]	Path string to Configuration file.

Returns:

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 1: 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
```

7.2 PS_DELETE_PSOBJECT

int ps_delete_psobject(const char * name)

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

Parameters:

name [in]	Previously created PowerSplitting name

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:

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

7.3 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. PowerSplitting 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

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
WS_OPENFAILED	Could not open the WaveShaper. May be a connection problem.

Example:

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

7.4 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 WaveShaper name

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

7.5 PS_LOAD_PSP

```
int  ws_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

Returns:

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

Error Code	Description
WS_SUCCESS	Command successfully executed
WS_WAVESHAPER_NOT_FOUND	PowerSplitting 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
WS_INVALIDPROFILE	Other parsing error
WS_OPENFAILED	Could not open the WaveShaper. May be a connection problem.
WS_WAVESHAPER_CMD_ERROR	Error response from WaveShaper. May be communication corruption.

Example:

```
/* hard coded PSP 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
```

7.6 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.

7.7 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 Power Split 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 APPENDIX A -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 APPENDIX B – FILE LIST

The WaveShaper API includes the following files:

File Name	Description
ws_api.h	Header file contains all function and type definitions.
ws_psapi.h	Header file containing all Power Splitting functions.
wsapi.dll (.so)	WaveShaper API dynamic link library.
wsapi.lib	Static import library to wsapi.dll (Windows only)

DLL dependencies:

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

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

10 APPENDIX C- SAMPLE CODE

10.1 C SAMPLE CODE: WS_LOAD_PROFILE

```
/** @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;
}
```

10.2 PYTHON SAMPLE CODE: WS_LOAD_PROFILE

```
#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)
```

10.3 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));
}
```

10.4 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)
```