Quick Start Manual

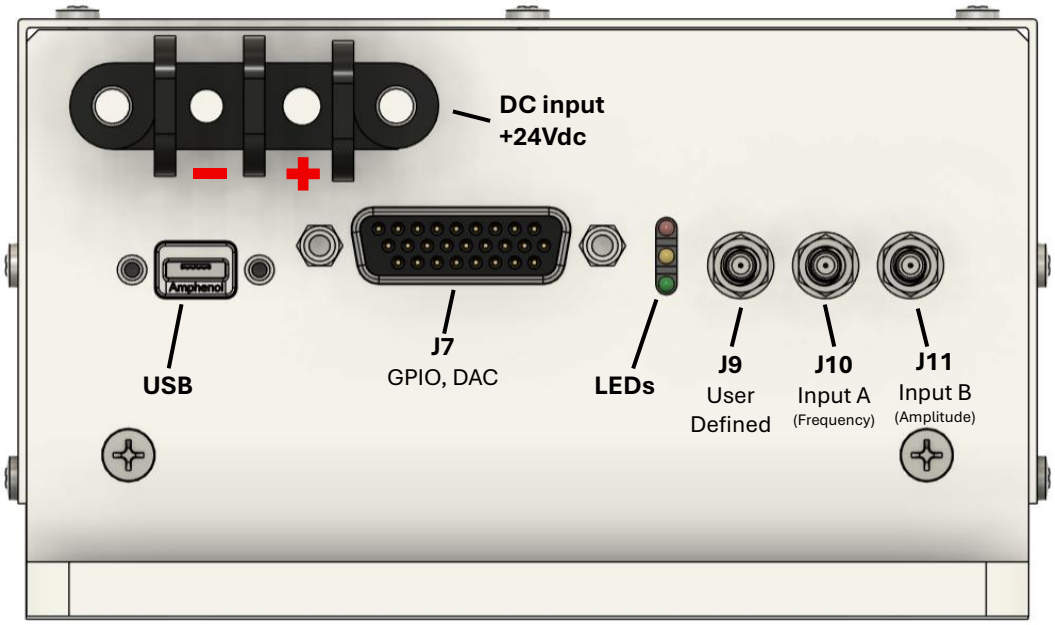
iVSA devices producing 15W RF

Contents

1. Hardware Overview 3
1.1 Front Panel Connections 4
1.2 Rear Panel Connections 4
1.3 Minimum Hardware Requirements 4
2. Software Installation and SDK Support 5
2.1 SDK Requirements 5
2.2 Python Installation 5
2.3 SDK Access 5
2.4 iVCS Control Panel 5
3. Power-Up and Interlock Behaviour 6
3.1 Initial Power-Up (No RF) 6
3.2 Interlock Function 6
3.3 DDS Amplitude Control (iVSA / iVCS) 6
4. Basic Functional Validation 7
4.1 Recommended Method 7
4.2 Validation Procedure 7
2.5 VCO Input Control 8
2.5.1 Conceptual Overview 8
2.5.2 Configuring Inputs 8
2.5.3 Storing the Mapping as the Default State 8
5. Compensation (Amplitude & Phase LUTs) 9
5.1 Purpose of Compensation 9
5.2 LUT Types 9
5.3 Creating and Applying Compensation Tables 9
5.4 Device-Specific Compensation Using AODevice 10
5.5 Storing compensation in non-volatile memory as default 12
6. Logging, Diagnostics and Support 13
6.1 Firmware and SDK Version Identification 13
6.2 Log Configuration 13
6.3 Locating Log Files 13
6.4 Support Guidance 13

1. Hardware Overview

Front



Rear

1.1 Front Panel Connections

Label	Connector	Function
DC in	Terminal Block (11mm)	DC supply
USB	USB-Micro	Primary communications interface to host PC (SDK / Control Panel)
J7	High-density D-Sub	GPIO, DAC, hardware gate, auxiliary control
LEDs	Tri-colour LED stack	Status indication (power, RF enable, interlock)
J9	SMB	User-defined analogue input (configurable via SDK)
J10	SMB	Input A – Frequency control (external analogue)
J11	SMB	Input B – Amplitude control (external analogue)

Note: By default, J9 is unassigned. Its function can be configured in firmware to map to supported SDK-controlled parameters.

1.2 Rear Panel Connections

Label	Connector	Function
RF1	BNC	RF Output Channel 1 (50 Ω)
RF2	BNC	RF Output Channel 2 (50 Ω)
AO INT	3-Pin Binder	Interlock interface controlling RF amplifier supply

Important: RF output must always be terminated into a **50 Ω load** during testing and operation.

1.3 Minimum Hardware Requirements

To safely power and communicate with the synthesiser, the following conditions must be met:

- DC power supply within datasheet limits
- USB-connected host PC
- iMS SDK **v2.0.5 or later**
- RF outputs terminated into 50 Ω
- Interlock loop satisfied (see Section 6)

2. Software Installation and SDK Support

2.1 SDK Requirements

SDK version **2.0.5** is required at minimum to interface with the iVSA range.

Note: Non-Python SDKs (C++ and C#) with version numbers **greater than 2.0** are expected to be released in **Q1 2026**.

2.2 Python Installation

Install the Python SDK bindings using:

```
pip install imslib
```

This installs the Python wrapper compatible with the Isomet iMS SDK.

2.3 SDK Access

- Official SDK: **www.isomet.com**
- Open-source reference implementation: **https://github.com/Isomet-Corporation**

Full, working examples are provided in the Isomet GitHub repositories.

2.4 iVCS Control Panel

Isomet has made available **iVCSControlPanel.exe** as a recommended starting point for validating the operation of devices. This file requires no dependencies such as python or imslib to be installed on the host machine to run.

3. Power-Up and Interlock Behaviour

3.1 Initial Power-Up (No RF)

Upon application of DC power:

- No interlock is required to communicate with the control unit
- The **green LED** will illuminate and pulse at approximately **1 Hz**
- The device becomes visible to the SDK and Control Panel software

At this stage, **RF output is disabled**.

3.2 Interlock Function

The interlock controls the DC supply to the internal RF amplifiers.

LED State	Meaning
Green (pulsing)	Device powered and active (does not indicate RF state)
Amber	RF enabled and active
Red	Interlock open / RF inhibited

Refer to the product datasheet for exact interlock pin definitions.

3.2.1 Enable Software Gate

In the iVSA range, RF output requires the interlock to be satisfied. One of these interlocks is a software gate and can be enabled using the provided below

```
import imslib
from imslib import SystemFunc

sysf = SystemFunc(ims)
sysf.EnableAmplifier(True)
```

The **hardware gate** input (J7 pin 17, TTL active high) must also be asserted for RF output.

3.3 DDS Amplitude Control (iVSA / iVCS)

On power-up, the iVSA / iVCS DDS output amplitude is initialised to a default value (50%). The DDS amplitude can be modified dynamically at runtime using the iMS SDK. We recommend beginning with the default values and increasing only if necessary.

3.3.1 Set DDS amplitude in software

```
from imslib import Percent, SignalPath

sp = SignalPath(ims)
sp.UpdateDDSPowerLevel(Percent(50.00))
```

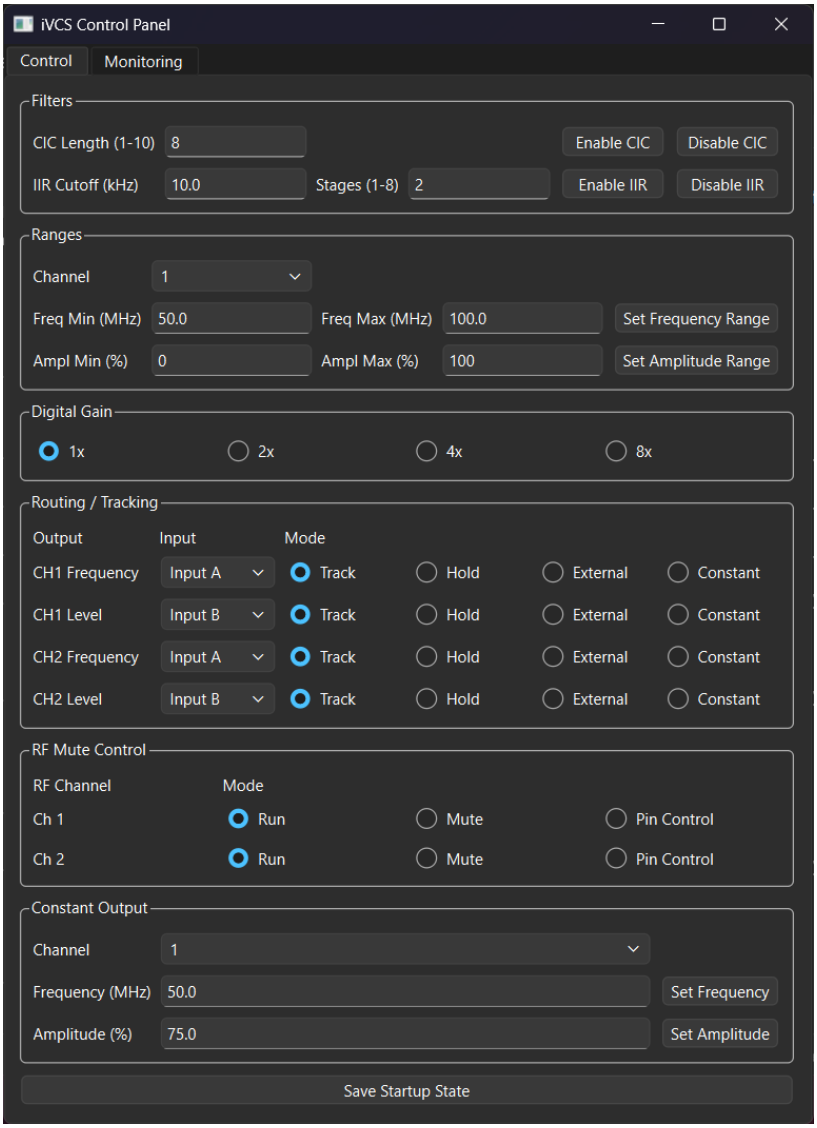
4. Basic Functional Validation

4.1 Recommended Method

Isomet strongly recommends using **iVCSControlPanel.exe** for first-time validation.

4.2 Validation Procedure

1. Connect RF outputs to 50 Ω loads
2. Apply DC power
3. Launch **iVCSControlPanel.exe**
4. Navigate to the **Constant Output** section
5. Set both channels to the device centre frequency
6. Enable interlock
7. Increase amplitude gradually
8. Verify RF output against factory test data



Filters
Advanced Setting; Controls the filtering stages of the devices fast ADC's

Ranges
Allows the user to define the mapping of the analogue input to output values.

Digital Gain
Advanced Setting; Defines the gain of the analogue inputs

Routing / Tracking
Mapping analogue inputs to the values they will control. Defines signal tracking mode

RF Mute Control
Define RF output mute mode per channel

Constant Output
Set a constant frequency and amplitude on any channel.

Save Startup State
Stores selected settings in non-volatile memory for later recall.

Note: iVCS control panel will not set software gate and so this must be done prior to the starting of this program.

2.5 VCO Input Control

This section explains how to explicitly map the external analogue inputs **J10** and **J11** to internal DDS **frequency** and **amplitude** control nodes using the iMS SDK signal-path routing interface. This configuration enables VCO-style analogue control of the synthesiser without the use of images or sequences.

2.5.1 Conceptual Overview

- **J10** (Input A) is typically mapped to **DDS frequency control**
- **J11** (Input B) is typically mapped to **DDS amplitude control**

The mapping is performed by routing an external input node to an internal VCO/DDS control output node using the `Route()` method.

2.5.2 Configuring Inputs

This can be done using the **iVCSControlPanel.exe** or through the python code shown below.

```
import imslib
from imslib import VCO, MHz, Percent

#First connect to device as shown in test01_scan.py
vco = VCO(ims)

#mappin input ranges
vco.SetFrequencyRange(MHz(80.0), MHz(120.0))
vco.SetAmplitudeRange(Percent(0.0), Percent(100.0))
# adjust to suit your application

# Route external analogue input A (J10) to CH1 frequency
ok = vco.Route(VCO.VCOOutput_CH1_FREQUENCY, VCO.VCOInput_A)
if not ok:
    raise RuntimeError("Route() failed for CH1 frequency")

# Route external analogue input B (J11) to CH1 amplitude
ok = vco.Route(VCO.VCOOutput_CH1_AMPLITUDE, VCO.VCOInput_B)
if not ok:
    raise RuntimeError("Route() failed for CH1 amplitude")
```

2.5.3 Storing the Mapping as the Default State

Once verified, the routing configuration can be stored so it is automatically restored on power-up using either. the **iVCSControlPanel.exe** or the python code shown below.

```
ok = vco.SaveStartupState()
if not ok:
    raise RuntimeError("SaveStartupState() failed")
```

5. Compensation (Amplitude & Phase LUTs)

This section describes how amplitude and phase compensation is generated, applied, verified, and stored on iVSA / iVCS devices.

5.1 Purpose of Compensation

Compensation allows linearisation of:

- RF amplitude
- RF phase
- Optional sync outputs

This is critical when driving **acousto-optic devices**, especially for:

- Constant diffraction efficiency
- XY deflector linearity
- Beam Steered applications

5.2 LUT Types

LUT Size	Scope	Use Case
~57 KB	Global	Same compensation applied to all channels (legacy devices)
~225 KB	Channel-specific	Independent channel compensation (recommended for XY systems)

5.3 Creating and Applying Compensation Tables

Use the following procedure to generate and apply compensation using the iMS SDK:

Compensation tables are generated in software using the iMS SDK and are typically derived from measured device performance (e.g. RF power, diffraction efficiency, or phase response versus frequency).

1. **Characterise the AO device** across the intended operating frequency range
2. **Generate a compensation table** in software that maps frequency to amplitude and/or phase corrections
3. **Download the table to the iVSA device** using the SDK
4. **Optionally store the table in long-term (non-volatile) memory** so it is restored automatically at power-up

Note: you may find AN0425 AO-BeamSteer and AN190811 Compensation LUTs a useful reference for method and procedure when generating .lut files.

5.4 Device-Specific Compensation Using AODevice

The SDK exposes the 'AODevice' class as a *shopfront* to an internal AO-device database and associated operating-parameter calculations.

An `AODevice` instance may be created:

- From a **model number**, which retrieves known material, centre frequency, bandwidth, wavelength range, and geometric constants from the internal database, or
- As a **hypothetical device**, by explicitly supplying:
    - Crystal material
    - Geometric constant
    - Centre frequency
    - Operating bandwidth

From an `AODevice`, the SDK can compute:

- External Bragg angle
- A physics-derived **compensation function** via `GetCompensationFunction()`

Note: We strongly recommend using this feature to generate a baseline table for **beam-steered** applications

5.4.1 – 'AODevice' example code

```
import imslib
from imslib import Percent, CompensationTable, CompensationFunction

#Create a compenstaion sized for the connected device
ctbl = CompensationTable(ims)

#set all amplitudes to 50% (recommended starting point)
for i in range(len(ctbl)):
    ctbl[i].Amplitude = Percent(50.0)

# Generate phase compensation from an AODevice model
aod_model = "D1384-aQ120L"   # <- must match a model in AODevice
aod = imslib.AODevice(aod_model)

cfunc_phase = aod.GetCompensationFunction()
cfunc_phase.PhaseInterpolationStyle = (
    CompensationFunction.InterpolationStyle_LINEXTEND
)

ok = ctbl.ApplyFunction(cfunc_phase,
imslib.CompensationFeature_PHASE)
if not ok:
    raise RuntimeError("ApplyFunction() failed")

# Download the compensation table to hardware (+ optional verify)
dl = imslib.CompensationTableDownload(ims, ctbl)

if not dl.StartDownload():
    raise RuntimeError("StartDownload() failed")

if not dl.StartVerify():
    raise RuntimeError("StartVerify() failed")
```

5.5 Storing compensation in non-volatile memory as default

Users may wish to store a compensation function in non-volatile memory as the default so that it is loaded to the device at startup with no requirement for software. This is done as shown in the following example.

Firmware / SDK Requirements:

- Minimum firmware version to support long-term storage: **v1.1.24**
- Minimum SDK version required: **v2.0.6**

5.5.1 – Example of saving to non-volatile memory

```
# setup signalpath, validate lut file and setup importer
sp = imslib.SignalPath(ims)

if os.path.isfile("YOUR_LUT_file.lut") == False:
    print("Can't find LUT File")
    sys.exit()

importer = imslib.CompensationTableImporter("YOUR_LUT_file.lut")
if not importer.IsValid():
    print("Error in LUT File.")
    sys.exit()

#retrieve global LUT and download to hardware
comp = importer.RetrieveGlobalLUT()
ctdl = imslib.CompensationTableDownload(ims, comp)

#store as default
default_name = "DEF_LUT"
idx = ctdl.Store(imslib.FileDefault_DEFAULT, default_name)
print(f"Stored LUT as DEFAULT startup table: '{default_name}' (FST
index={idx})")
```

6. Logging, Diagnostics and Support

6.1 Firmware and SDK Version Identification

To retrieve version information programmatically:

- **Firmware Version**
  - `IMSController.GetVersion()`
- **SDK Version**
  - `LibVersion.GetVersion()`

This information should be included in all support requests.

6.2 Log Configuration

Log configuration file:

`C:\Users\<username>\AppData\Local\Isomet\iMS_SDK\logging.ini`

Key parameters:

- `DisableLogging = false`
- Severity filters:
  - Error: `%Severity% >= error`
  - Trace: `%Severity% >= trace`

6.3 Locating Log Files

Log files are stored in:

`C:\Users\<username>\AppData\Local\Temp\`

File naming format:

`imslog_00000.log`

The index increments with each new session.

6.4 Support Guidance

When contacting Isomet support, please provide:

- Device model and serial number
- Firmware version
- SDK version
- Relevant log files
- Description of RF load and operating conditions