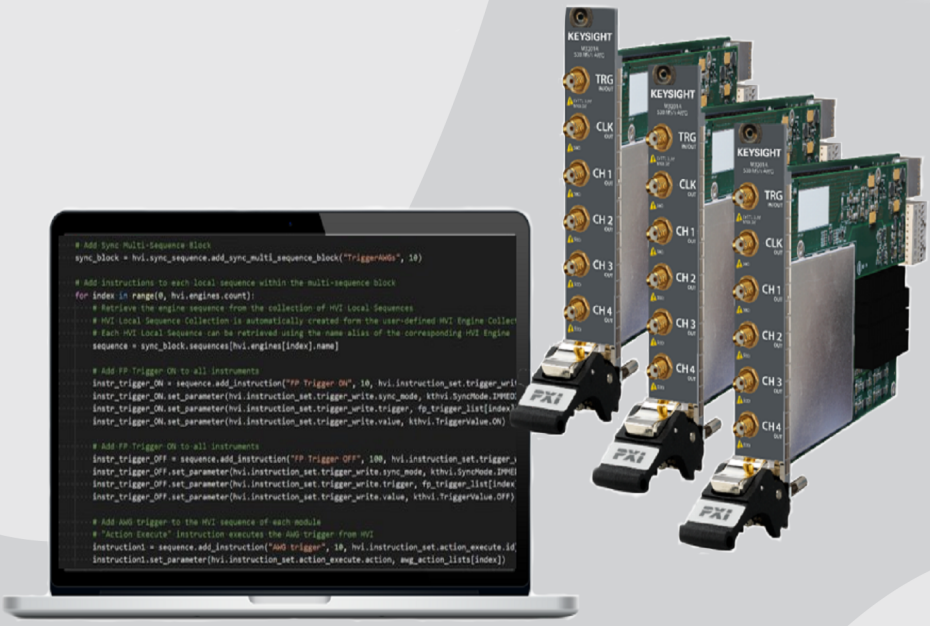
PathWave
Test Sync Executive 2023C

System Setup Guide



KEYSIGHT

SYSTEM SETUP GUIDE

Table of Contents

System Setup Guide

This guide describes how to set up a single or multi-chassis system using Keysight PXIe Chassis and PXIe System Synchronization Modules. It also describes how to set a reference clock and how to configure everything in your software code by using the TSE API delivered by PathWave Test Sync Executive.

1. Software Install

For your setup to function correctly, you must ensure all the relevant instrument drivers and software components are installed in your system. This includes:

- PathWave Test Sync Executive.
- Chassis drivers and firmware.
- Instruments drivers and firmware.
- System Synchronization Modules (SSM) drivers and firmware, if present.
- *High Performance Reference Clock Source* (HPCRS) option drivers, if present.

If you intend to use more than 6 chassis or require remote application execution, you also require the PathWave Test Sync Executive options:

- TSE Service.
- Keysight Distributed Infrastructure (KDI).

The full list of instruments supported by PathWave Test Sync Executive along with links to driver download pages is located at Instrument and Chassis Software and Firmware Requirements for KS2201A.

For full installation instructions see the relevant documentation for each component.

2. Keysight System Synchronization Modules (SSM)

KS2201A PathWave Test Sync Executive includes multi-chassis topologies that use the Keysight M9032A/M9033A PXIe *System Synch ronization Modules* (SSMs). The previous means of interconnecting multiple PXI chassis using M9031A modules was discontinued starting from the KS2201A 2022 release. Compared to the discontinued M9031A module, the SSM has a much wider range of functions including:

- Distribution of a precise reference clock.
- Management of *Fast Data Sharing* (FDS).
- Chassis interconnectivity.
- Synchronization of all the PXI instruments in the multi-chassis.

2.1. M9032A and M9033A PXIe SSM Overview

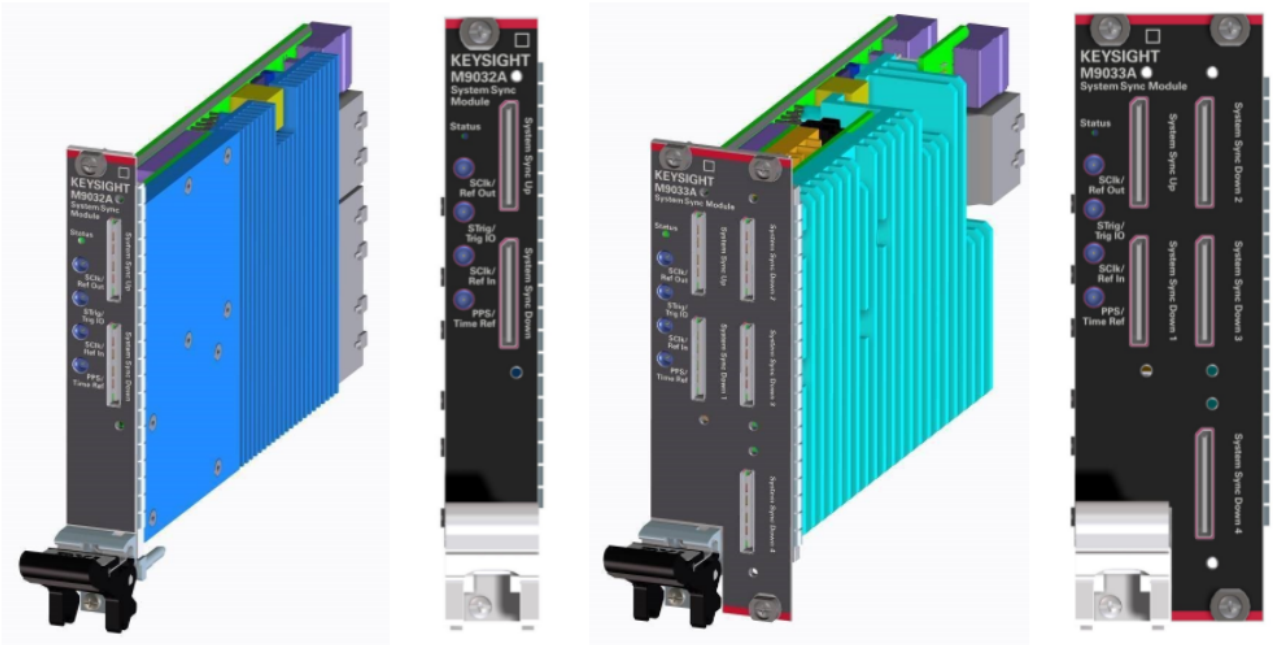
The M9032A/M9033A are PXIe *System Synchronization Modules* (SSM). These include an onboard high-quality 10MHz Oven Controlled Crystal Oscillator (OCXO) to achieve a very precise synchronization among various measurement instruments distributed across different chassis. The M9032A/M9033A System Synchronization Module functionalities can only be successfully deployed on chassis compliant with the *PXI-Express* (PXIe) standard. The SSM must be inserted in the timing slot of the PXIe chassis.

Keysight PXIe System Synchronization Module is available in two form factors, which only differ in their connectivity capabilities:

- M9032A is a one-slot PXIe System Synchronization Module with 1 System Sync Upstream and 1 System Sync Downstream ports.
- M9033A is a two-slot PXIe System Synchronization Module with 1 System Sync Upstream and 4 System Sync Downstream ports.

For further information about these SSMs including detailed performance specifications, see the M9032A/M9033A User's Guide, available at Keysight PXI Products.

The following image shows the physical M9032A and M9033A SSMs:



2.2. M9032A and M9033A Connectivity

2.2.1. Front Panel

The M9032A and M9033A Front Panel contains various connectors that can be used for both multi-chassis interconnection and configuration of the reference clock source.

Front Panel *Sub Miniature Push-on* (SMP) IOs

Front Panel (FP) SMP connectors are:

SClk/Ref Out:

Outputs a copy of the system clock or a reference clock signal.

STrig/Trig IO:

Receives an arbitrary trigger signal.

SClk/Ref In:

Receives the reference clock signal.

PPS/Time Ref:

Receives a Pulse Per Second (PPS) signal.

The front panel SMP connectors can be used to share input and output reference clocks.

2.2.2. System Sync ports

System Sync ports use PCIe *Optical Copper Link* (OCuLink) connectors. System Sync ports are used for chassis interconnection and synchronization in the multi-chassis system. The signals in the System Sync include:

- Clocking (System Sync only).
- Triggering.
- Data.

The different SSM models have the following front panel System Sync ports:

The M9032A has 2 System Sync ports:

- 1 System Sync Upstream.
- 1 System Sync Downstream.

The M9033A has 5 System Sync ports:

- 1 System Sync Upstream.
- 4 System Sync Downstream.

Each System Sync Downstream port can connect to the System Sync Upstream port of another SSM placed in a different chassis. For more information, see the section below about Inter/Intra-chassis Connectivity.

2.2.3. PXIe Backplane DSTAR Connectivity

The M9032A and M9033A are placed in the Timing Slot of a PXIe chassis which enables them to support the DSTAR connectivity built into the chassis.

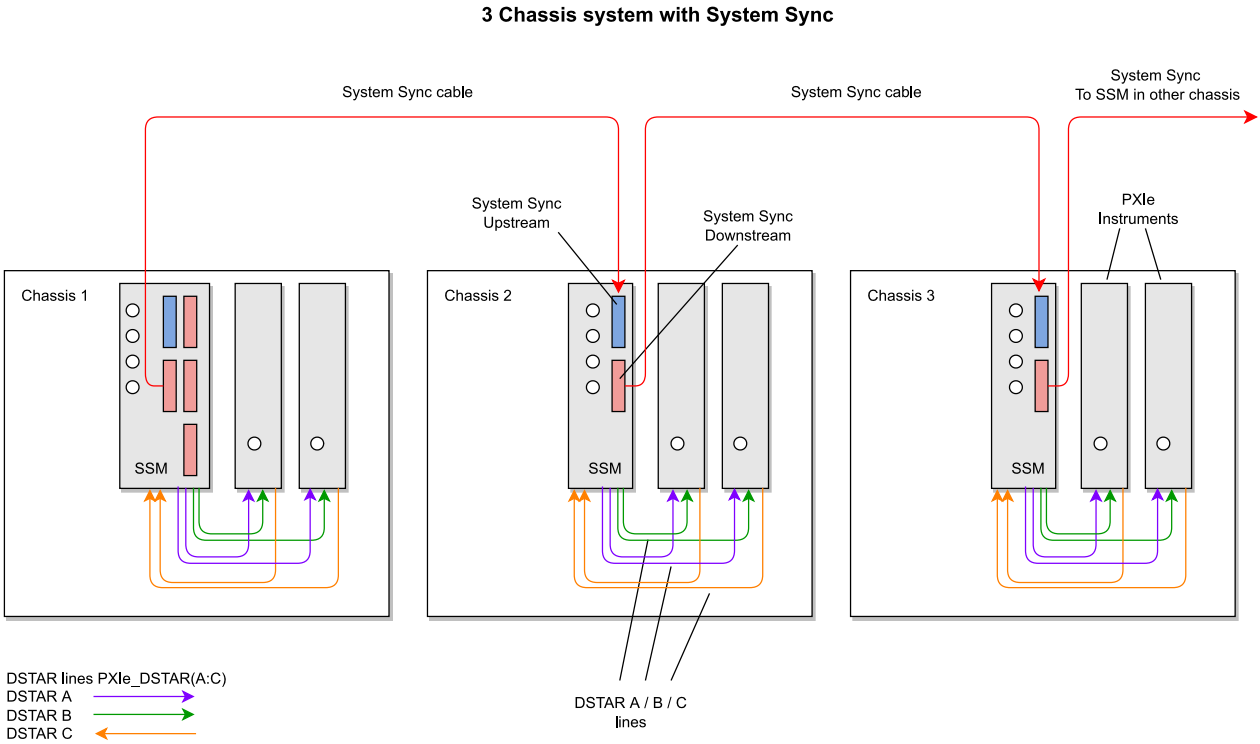
DSTARA/B/C are multi-instrument point to point connections inside a chassis. DSTARA is used to carry the clock signal. DSTARB and DSTARC carry trigger or data signals.

2.3. Inter/Intra-chassis connectivity, Synchronization and Data-Sharing Functionalities

An SSM can enable both multi-chassis and multi-instrument interconnections. With these connections, SSMs enable synchronization and data sharing across all the instruments in a multi-chassis system.

- Multi-chassis interconnections are made with System Sync connections using their capability to interconnect two SSMs together through their System Sync Downstream/Upstream ports.
- Intra-chassis, multi-instrument interconnections are made with PXIe DSTARA/B/C connections. The SSM can share the precise reference clock over the DSTARA signal.

The following diagram shows a 3 chassis system connected with System Sync cables and DSTARA/B/C signals in each chassis:



Data can be shared across System Sync and DSTAR connections in several different ways:

- The reference clock can be shared between two interconnected SSMs using the System Sync connection between System Sync Downstream/Upstream ports.

- The System Sync connection can share the signals sent over the PXI_TRIG[0:7] trigger buses, from one SSM to the next. This enables the SSMs to share PXI sync resources used by PathWave Test Sync Executive for the *Hard Virtual Instrument* (HVI) across the different chassis.
- System Sync connections can route data shared using *Fast Data Sharing* (FDS) between PXIe instruments.
- The SSM can send the data between two modules located in the same chassis using the DSTARB/C signals.
- Data can be sent through the System Sync connections to route it to instruments located in a different chassis.

3. Configuring a System with SSMs and System Sync Connectivity

3.1. Multi-chassis Configuration

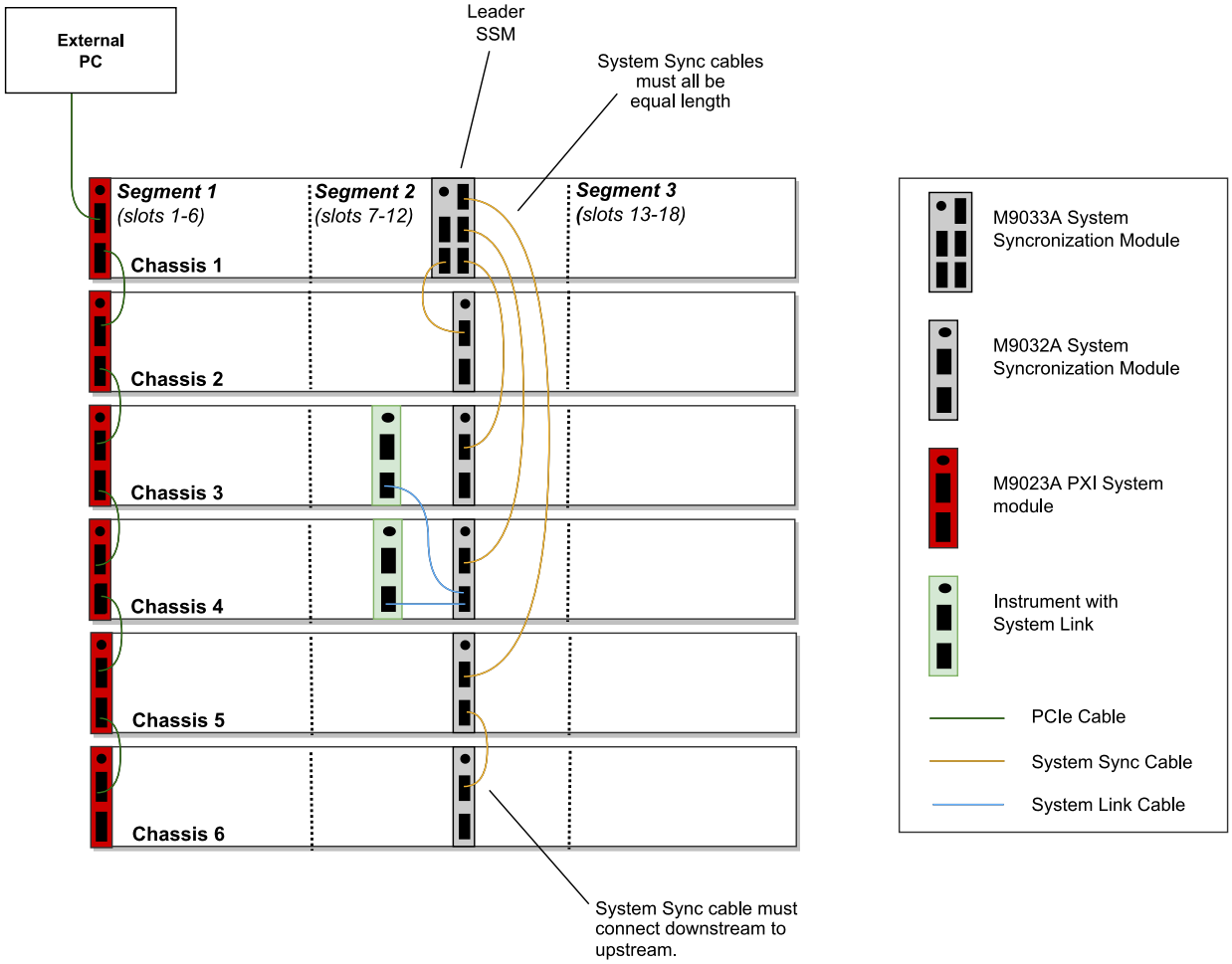
In a multi-chassis system connected with Keysight PXIe System Synchronization Modules, you must include one SSM in each chassis that is part of the system. Each SSM must be inserted in the **timing slot** of your chassis. This is typically slot 10 in Keysight 18-slot chassis, but it can be a different slot number in different chassis models. The SSMs are connected to each other with System Sync cables.

One SSM is automatically chosen as a leader and it is used to synchronize all the instruments in the multi-chassis system. The SSM chosen as leader is the SSM that has no incoming connection to its System Sync Upstream port. The leader SSM distributes a replica of the reference clock signal to the SSMs located in the other chassis. It does this through point-to-point connections between System Sync Downstream/Upstream ports. In the example multi-chassis system shown in the following diagram, the leader SSM is in Chassis 1.

A multi-chassis PXIe system may be configured to use many different reference options. For a list of those options and descriptions of how to configure them, see the section *Clocking* in this document. For one of those reference options, an SSM is chosen as a leader and uses its internal Oven Controlled Crystal Oscillator (OCXO) clock to synchronize all the instruments included in the multi-chassis system.

NOTE A Multi-chassis system based on the older M9031A modules required an external reference clock generator to distribute the precise common 10 MHz reference clock signal across different chassis. In the multi-chassis topology delivered by PathWave Test Sync Executive, the SSM assumes the function of the **reference clock signal generator/distributor**, by sharing a reference clock generated by an internal PLL. This PLL can be fed by different sources (as explained later in this document) including the OCXO inside the SSM, which generates a 10 MHz sine wave. An external 10 or 100 MHz reference signal can still be connected to the SSM SClk / Ref Input port, to sync it together with other clusters.

The following diagram shows an example of a 6 chassis system connected with SSMs. In this system an M9033A SSM in chassis 1 distributes the reference clock to four M9032A SSMs located in each of the other chassis. The SSM in chassis 5 also forwards the clock to a sixth chassis.



This following code shows how to use the HVI Python API to define and use the SSMs in the multi-chassis system shown in the diagram. Each System Sync Downstream port connects to the System Sync Upstream port of another System Sync Module in a different chassis.

The first step is to define the SSMs placed in each of the chassis during the System Definition phase.

```
# Create SystemDefinition object
my_system = kthvi.SystemDefinition("MySystem")
#
# Define System Sync Modules
resource_id_ssm_1 = 'PXI0::CHASSIS1::SLOT10::INSTR'
resource_id_ssm_2 = 'PXI0::CHASSIS2::SLOT10::INSTR'
resource_id_ssm_3 = 'PXI0::CHASSIS3::SLOT10::INSTR'
resource_id_ssm_4 = 'PXI0::CHASSIS4::SLOT10::INSTR'
resource_id_ssm_5 = 'PXI0::CHASSIS5::SLOT10::INSTR'
resource_id_ssm_6 = 'PXI0::CHASSIS6::SLOT10::INSTR'
#
# In the options, SSMs are set to be simulated with Simulate=true and there are a number of parameters.
# For the hardware SSM instruments, set options to an empty string.
options1 = "Simulate=true,DriverSetup=Model=M9033A"
options2 = "Simulate=true,DriverSetup=Model=M9032A"
options3 = "Simulate=true,DriverSetup=Model=M9032A"
options4 = "Simulate=true,DriverSetup=Model=M9032A"
options5 = "Simulate=true,DriverSetup=Model=M9032A"
options6 = "Simulate=true,DriverSetup=Model=M9032A"
#
sync_module_1 = my_system.interconnects.add_sync_module(resource_id_ssm_1, options1)
sync_module_2 = my_system.interconnects.add_sync_module(resource_id_ssm_2, options2)
sync_module_3 = my_system.interconnects.add_sync_module(resource_id_ssm_3, options3)
sync_module_4 = my_system.interconnects.add_sync_module(resource_id_ssm_4, options4)
sync_module_5 = my_system.interconnects.add_sync_module(resource_id_ssm_5, options5)
sync_module_6 = my_system.interconnects.add_sync_module(resource_id_ssm_6, options6)
```

NOTE In the HVI System Definition phase, the SSMs are added to the interconnects collection by using their resource ID and options. Same as for the chassis, it is not necessary to open objects representing the SSMs that are included in the multi-chassis system.

The next step is to define the interconnections among the System Sync Downstream/Upstream ports of each pair of SSMs. The SSM System Sync ports can only be connected Downstream to Upstream.

```
# Define connections among System Sync connectors of the SSMs
#
# Connect SSM 1 to SSM 2
ssm1_downstream_sync1 = sync_module_1.connectivity.systemsync_downstream[0]
ssm2_upstream_sync    = sync_module_2.connectivity.systemsync_upstream[0]
ssm1_downstream_sync1.set_connection(ssm2_upstream_sync)
#
# Connect SSM 1 to SSM 3
ssm1_downstream_sync2 = sync_module_1.connectivity.systemsync_downstream[1]
ssm3_upstream_sync    = sync_module_3.connectivity.systemsync_upstream[0]
ssm1_downstream_sync2.set_connection(ssm3_upstream_sync)
#
# Connect SSM 1 to SSM 4
ssm1_downstream_sync3 = sync_module_1.connectivity.systemsync_downstream[2]
ssm4_upstream_sync    = sync_module_4.connectivity.systemsync_upstream[0]
ssm1_downstream_sync3.set_connection(ssm4_upstream_sync)
#
# Connect SSM 1 to SSM 5
ssm1_downstream_sync4 = sync_module_1.connectivity.systemsync_downstream[3]
ssm5_upstream_sync    = sync_module_5.connectivity.systemsync_upstream[0]
ssm1_downstream_sync4.set_connection(ssm5_upstream_sync)
#
# Connect SSM 5 to SSM 6
ssm5_downstream_sync = sync_module_5.connectivity.systemsync_downstream[0]
ssm6_upstream_sync    = sync_module_6.connectivity.systemsync_upstream[0]
ssm5_downstream_sync.set_connection(ssm6_upstream_sync)
```

3.2. Chassis Supported for Multi-Chassis Systems

The following Keysight chassis models are supported:

- M9018B
- M9019A
- M9046A

Software and firmware version requirements are listed on-line here: Chassis Software and Firmware Requirements for KS2201A .

NOTE If you mix different chassis models in your multi-chassis setup, you may observe some skew across the different chassis and different performance depending on the different chassis characteristics.

Non Keysight chassis are not supported for multi-chassis systems.

4. Clocking

4.1. Clock Types

In a single or multi-chassis system there are 4 types of clocks used for synchronization and instrument-related tasks:

- Reference clock.
- System clocks.
- Analog clocks.
- Sample clocks.

All these clocks are synchronous with one another, but are used for different purposes and can be configured in different ways trading off performance and complexity/cost.

4.1.1. Reference Clock

The Reference clock used as reference for generation of other clocks, it determines the absolute frequency and lowest-frequency offset phase noise performance of the analog instrumentation's inputs and outputs. That is because all of the other clocks are phase-locked to the Reference Clock. A PXIe system can either use its own internal reference clock or phase-lock to an external reference clock. It can also provide external reference clock outputs for other instrumentation to phase-lock to.

4.1.2. System Clocks

The relevant clocks that drive the internal logic of individual instruments. These include clocks automatically reported to HVI and clocks users reports explicitly using the TSE API. The System clocks synchronize all the digital operation of all instruments and the PXIe platform. These clocks are derived from the Reference Clock and are used by, for example, the PathWave FPGAs Sandbox logic, the HVI Engine core clock, Fast Data Sharing and other digital capabilities in the instruments. Basically, a system clock is clock that is neither the reference clock nor an analog clock.

4.1.3. Analog Clocks

Clock responsible for generating the analogue signals in an instrument, the analog signal specifications such as phase noise, depend on the specs of this clock. The Analog Clocks are intermediate frequency clocks from which the instrument's Sample Clocks are derived. Like the Sample clocks, the Analog Clocks affect the overall phase noise performance and skew drift of the instrument analog inputs and outputs. In the simplest clock configurations, each peripheral module generates it's own independent Analog Clock. In the highest fidelity clock configuration, a single common Analog clock is generated by the *High Performance Reference Clock Source* (HPRCS) and is distributed to all the individual peripheral modules though external cables and power dividers.

4.1.4. Sample Clocks

The instrument's ADCs and DACs that digitize analog input signals and generate analog output signals are clocked by their own internal Sample Clocks. The various types of peripheral modules use different sample clock frequencies even though they are ultimately derived from the same Reference clock. These sample clocks determine the overall phase noise performance and skew drift of the analog inputs and outputs because they directly clock the instrument's ADCs and DACs.

4.2. System Clock Distribution using SSM and System Sync connectivity

In a multi-chassis system based on the Keysight PXIe SSMs and chassis, the SSM with no other SSM connected to its System Sync Upstream port acts as the leader. This leader SSM forwards a copy of the system clock to other SSMs using System Sync cables. In turn, each SSM shares the forwarded system clock with the instruments located in their respective chassis using the PXIe DSTARA backplane signal.

NOTE You are not required to set the the leader in the TSE API. The leader SSM is determined by the hardware connections. That is, the leader role is automatically taken by the SSM that has no System Sync cable connected to its System Sync Upstream port.

4.3. Enabling chassis clock outputs

If you are using a clock output from a chassis you can enable it in the TSE API.

The chassis clock outputs are available in the chassis and you can access them by their name as follows:

```
# Get the Clock configuration for the Rear Panel 10MHz output port from the Chassis
ktHvi.SystemDefinition definition("Name")
#
chassis = definition.add_chassis(1)
#
clockOutputRp10Mhz = chassis.clock_outputs["RP10MHzOut"]
clockOutputRP10Mhz.set_enabled(true/false)
```

Some clock outputs support one single frequency and others support multiple frequencies. For the outputs supporting only one frequency, no frequency must be provided when enabling/disabling them. If the clock outputs do support multiple frequencies, you must specify what frequency (in Hz) you want to enable.

When you disable the clock, the frequency argument is ignored.

The following code shows some examples and error cases:

```
clockOutputRp10Mhz = chassis.clock_outputs["RP10MHzOut"]
clockOutputRP10Mhz.set_enabled(true) # Ok
#
clockOutputFpRef2Out = chassis.clock_outputs["FPRef2Out"]
clockOutputFpRef2Out.set_enabled(true, 10e6) # Ok
```

4.4. Overview of Supported Clocking Schemes

There are several possible different clocking configurations, the one you should use depends on the hardware and the application requirements. Some of the key aspects to consider when selecting a clocking scheme are:

1. System and Analog clock sources. The source for the System and intermediate-frequency analog clocks is a critical element that determines the system synchronization, phase noise and drift performance. The clock sources covered in this section include:
   a. PXIe chassis.
   b. System Sync Module.
   c. PXIe Chassis with *High Performance Reference Clock Source* (HPRCS). This is only available on Keysight PXIe chassis models M904xA.

2. Internal/external Reference clock. The clock that serves as reference for the System/Analog clocks can be generated internally by the selected source, or externally provided by the user, generated by a clock source external to the PXIe system. In systems that include the *High Performance Reference Clock Source* (HPRCS), and other external instrumentation that you wish to share a common Reference Clock, the best overall jitter performance will usually be achieved by phase-locking the other external instrumentation to the HPRCS Reference Clock instead of the other way around. If the overall system needs to be phase-locked to a GPS or atomic standard reference, you should phase-lock the HPRCS to the GPS or atomic standard and phase-lock all the other instrumentation to the HPRCS Reference Clock.
3. Instruments internal/external Analog Clock. Most instruments can either use an external Analog Clock or generate their own Analog Clock internally for convenience, however, using a common external Analog Clock will always provide the best performance because all peripheral module sample clocks will jitter and drift together.

The following tables show the different supported/recommended clocking schemes. We differentiate two different clocking architecture depending whether External Analog Clock distribution is used or instruments rely on the general-purpose reference to generate internally the Analog Clocks required to driver the RF/Analog outputs. For instruments that support both the use of a dedicated high performance Analog Clock reference distribution offers better performance.

4.4.1. Clocking schemes without External Analog Clock distribution

Clocking Scheme	Reference Clock Source	Description	Performance
A: Single-chassis w/o SSM.	Chassis: Internal 10MHz	An OCXO inside the chassis generates a 10 MHz reference clock. Independent Analog clocks are generated in each peripheral module.	See the chassis datasheet for exact phase noise performance. See the M5xxx PXIe instrument documentation for exact performance of channel for channel skew, jitter, and drift.
	Chassis: External 10MHz	The external reference clock must have a frequency of 10 MHz. As an example, it can come from a *Device Under Test* (DUT), another instrument that is part of the setup, etc. Independent Analog clocks are generated in each peripheral module.	-
B: Single/multiple chassis w/SSMs.	SSM: Internal 10MHz	An OCXO inside the SSM generates a 10 MHz reference clock. Independent Analog clocks are generated in each peripheral module.	See the SSM datasheet for exact phase noise performance. See the M5xxx PXIe instrument documentation for exact performance of channel for channel skew, jitter, and drift.
	SSM: External 10/100MHz	The external reference clock can have a 10 or 100 MHz frequency. As an example, it can come from a DUT, from another instrument that is part of the setup, etc. Independent Analog clocks are generated in each peripheral module.	-

4.4.2. Clocking schemes with External Analog Clock distribution using M904xA chassis

Clocking Scheme	Reference Clock Source	Description	Performance
C: Single/multiple chassis (no HPRCS) with External Analog clocks and SSMs	Chassis: Internal 10MHz	An OCXO inside the chassis generates a 10 MHz reference clock. A common Analog clock is externally distributed to each peripheral module.	See the chassis datasheet for exact phase noise performance. See the M5xxx PXIe instrument documentation for exact performance of channel for channel skew, jitter, and drift.
	Chassis: External 10MHz	The external reference clock must have a frequency of 10 MHz. As an example, it can come from a DUT, another instrument that is part of the setup, etc. A common Analog clock is externally distributed to each peripheral module.	-
D: Single/multiple chassis with HPRCS, External Analog clocks and SSMs.	HPRCS: Internal 10MHz	The HPRCS generates a 2.4 GHz sine wave that gets divided in frequency to generate a 100 MHz reference clock signal. A common Analog clock is externally distributed to each peripheral module.	This option provides the best performance in terms of phase noise. For more information, see the *Keysight PXIe Chassis M9046A Datasheet*, available at Keysight PXI chassis.
	HPRCS: External 10/100MHz	The external reference clock for the HPRCS can have a 10 or 100 MHz frequency. As an example, it can come from a DUT or another instrument that is part of the setup, etc. A common Analog clock is externally distributed to each peripheral module.	-

5. Clocking Configurations without External Analog Clock distribution

5.1. Single-chassis w/o SSM (Scheme A)

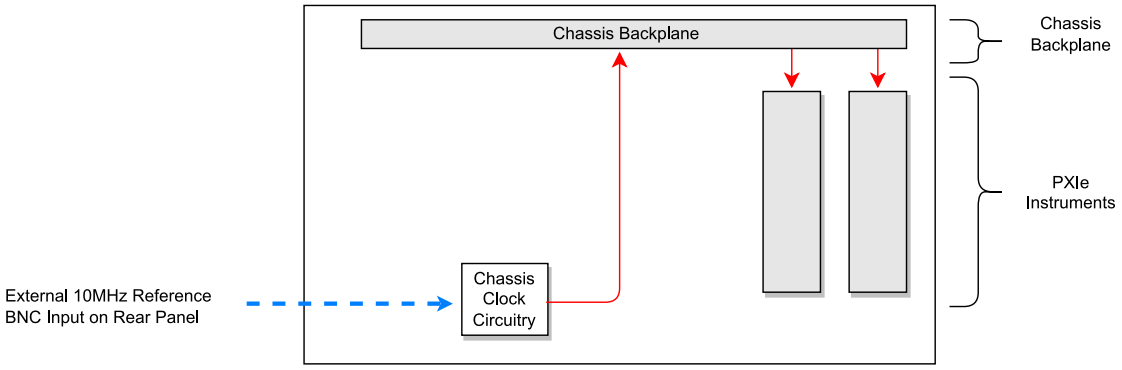
This is the simplest configuration and is the default if you have not specified another.

The chassis is the clock source. For the reference clock, there are two options:

1. **Internal (default):** This is the 10MHz clock built into the chassis (VCXO for the M9019A or OCXO for the M904xA).
2. **External:** A 10MHz signal connected to the 10MHz Ref BNC input located on the chassis rear panel.

The following diagram shows a chassis with an internal clock source (chassis clock) or an external clock source (blue):

Single chassis, no SSM
Internal or External Reference clock



All chassis have on their rear panel a 10MHz reference BNC input and a 10MHz reference BNC output. In the case of the M904x chassis, there are two Reference clock SMA outputs on the front panel.

This clocking scheme is rather constrained in terms of features because it only allows for a single chassis and, given that there is no SSM, advanced features like Fast Data Sharing are not available.

The following snippet shows how to configure the chassis as the clock source:

```
# Create SystemDefinition object
my_system = kthvi.SystemDefinition("MySystem")
#
# Define chassis
chassis = my_system.add_chassis(1)
#
# Select the chassis as ref. clock source
clockSource = chassis.clock_source
#
# Set the chassis as clock source
systemDefinition.clocking.reference_source = clockSource
#
# Explicitly set the clock source to use the internal OCXO as the reference clock (this is the default)
clockSource.set_mode(keysight_tse.ClockingReferenceMode.INTERNAL)
#
# Alternatively you can configure the chassis to use the external clock reference with the 10Mhz frequency
value in Hz
clockSource.set_mode(keysight_tse.ClockingReferenceMode.EXTERNAL, 10e6)
```

5.2. Single/multiple chassis w/SSMs (Scheme B)

Each SSM is equipped with an onboard high-quality 10MHz Oven Controlled Crystal Oscillator (OCXO) that can be used as the reference clock.

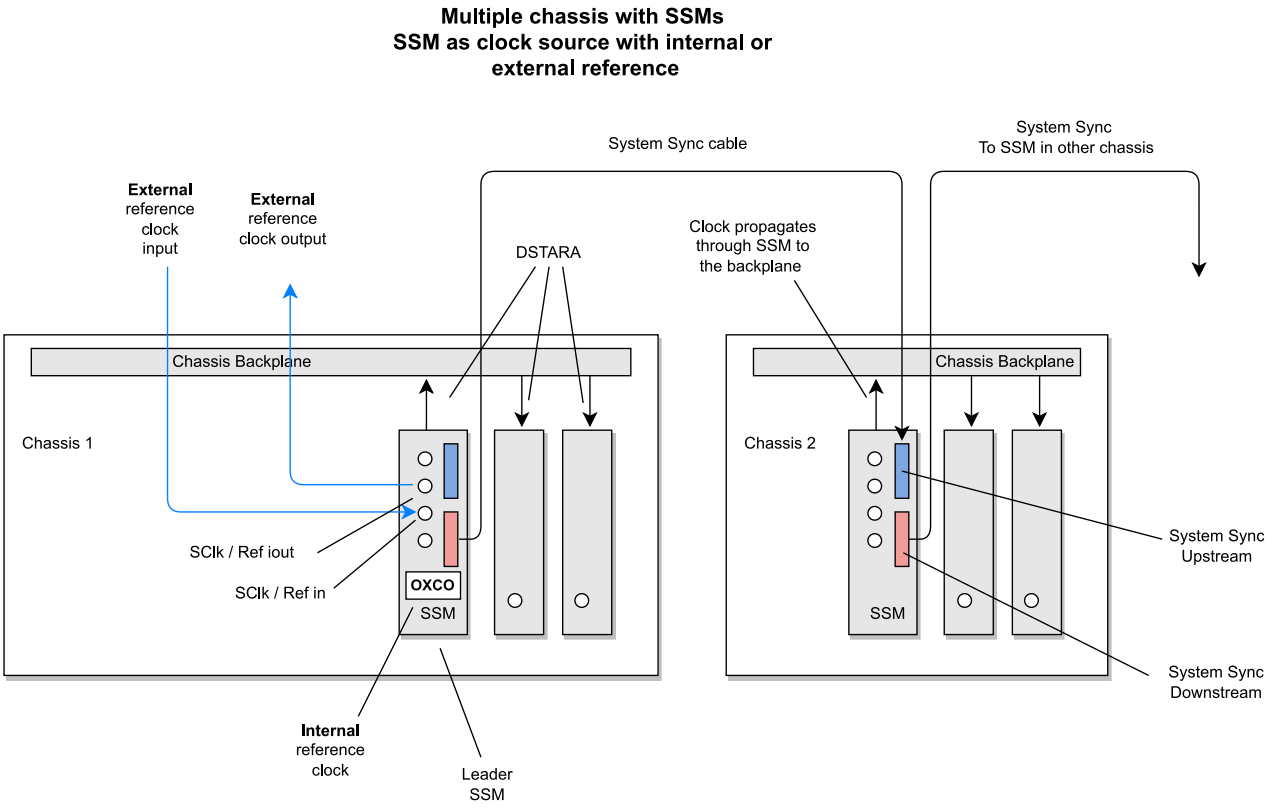
Alternatively, the chassis backplane reference clock output (Scheme A.2) or the optional *High Performance Reference Clock Source* (HPRCS) output (Scheme C) can be used as reference clock. The HPRCS option requires a Keysight PXIe Chassis model M9046A. Clocking scheme B assumes the external reference clock is neither output by the chassis nor by the HPRCS because otherwise further configurations would be required for proper operation.

The reference clock can be chosen from two options:

1. Internal: This is the default mode. The internal OCXO of the leader SSM is used as the reference clock.
2. External: An 10 MHz or 100 MHz external reference clock is connected to the SSM's front-panel SClk/Ref In SMP input.

The reference clock gets propagated to all the PXIe instruments within the same chassis through the DSTARA signal path. It gets propagated to the next SSM through the System Sync cable from the downstream connection on leader SSM to the upstream connection on the follower.

The following diagram shows the operation of the Clocking Scheme B. The clock is generated in the SSM in chassis 1 and is passed to the other instruments in the chassis via the DSTARA signal path in the backplane (red arrows). It is also passed to the next chassis via the System Sync cable (in black) where it propagates via the SSM in that chassis. The internal reference is the SSM's OCXO, and the external reference is shown in blue:



5.2.1. Configuring the SSM as the System Clock source

By default, if you do not specify anything, PathWave Test Sync Executive configures the leader SSM as the reference clock source using its internal OCXO clock. The leader SSM is defined by the hardware connections. In the TSE API, no additional definition other than the connections between SSM is required to identify the leader SSM. You must ensure the connections you define in software match the physical hardware connections between SSMs.

The following code shows how to configure a pair of chassis with SSMs where the OCXO clock is the reference clock source, `options` is set to an empty string:

```
# Create SystemDefinition object
my_system = kthvi.SystemDefinition("MySystem")
#
# Define all necessary follower SSMs depending the number of chassis
leader_ssm = my_system.interconnects.add_sync_module(SSM_1, options)
my_system.interconnects.add_sync_module(SSM_2, options)
#
# Define chassis
my_system.add_chassis(1)
my_system.add_chassis(2)
#
#  Select the leader SSM as ref. clock source
clockSource = interconnects[0].clock_source
#
# Set the SSM clock source
```

```
systemDefinition.clocking.reference_source = clockSource
#
# Explicitly set the clock source to use the internal OCXO as the reference clock (this is the default)
clockSource.set_mode(keysight_tse.ClockingReferenceMode.INTERNAL)
```

5.2.2. Configuring the SSM to explicitly use internal OCXO or external reference clock

The SSM leading the synchronization by default with its internal reference clock, can optionally be connected to an external reference clock. The external reference can come from, for example, a DUT or another source such as a PXIe frequency reference.

To use an external reference clock, you must:

- Connect the external reference source to the SSM's SClk / Ref in port.
- In the TSE API you must set the SSM to synchronize to an external reference clock. To do this, set the mode to `EXTERNAL` and set the frequency in Hz.

To use the external reference, change the final line in the previous code snippet to:

```
# Set clock mode to EXTERNAL and set frequency to 10MHz
clockSource.set_mode(keysight_tse.ClockingReferenceMode.EXTERNAL, 10e6)
```

6. Clocking Configurations with External Analog Clock distribution

Some instruments such as the analog ones in the Keysight M5xxx PXIe family derive their Sample Clocks from an Analog Clock source and perform best when configured to use an externally distributed Analog Clock. This section explains how to distribute the analog clocks to these and similar instruments.

The analog clock can be generated from either the M9546A HPRCS or from the M9046A chassis backplane board. The preferred choice for the analog clock source is the M9546A HPRCS inside a Keysight M9046A PXIe chassis because of it's superior phase noise. The HPRCS can generate a sine wave with frequencies of 2.4, 4.8, 9.6, or 19.2 GHz. In this section we assume the analog clock source being set to generate 2.4 GHz, because this is the frequency required by the Keysight M5xxx PXIe family. The frequency can be configured at purchase by choosing the corresponding option for the Keysight M9046A PXIe chassis. For more information, see the *Keysight PXIe Chassis M9046A User Manual* available at Keysight PXI chassis.

6.1. Chassis options for Analog clock generation and distribution

The following table lists the options available:

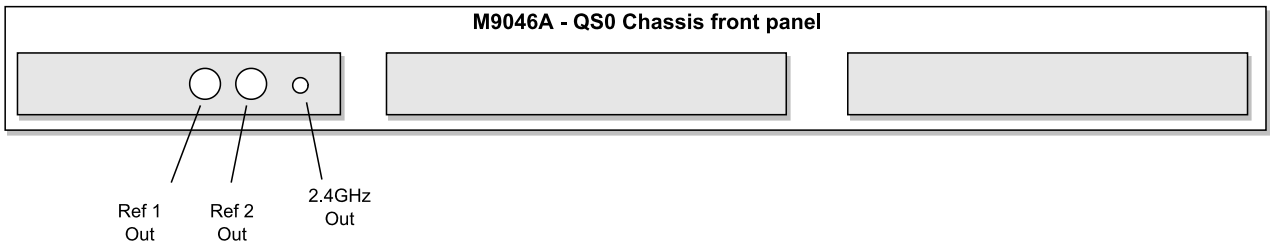
Source	Analog clock, locked to the reference clock	Performance
M9046A chassis with M9546A HPRCS	2.4, 4.8, 9.6, or 19.2 GHz	Best
M9046A chassis without M9546A HPRCS	2.4 GHz	Medium

6.2. M9046A Front Panel Clocking IO overview

The following diagrams show the M9046A chassis front panels and how they are connected in different configurations. The type and number of front panel connectors depend on the purchased hardware option for splitters and HPRCS: (-QS0, -QS1/3, -QS2). In the diagrams a frequency of 2.4 GHz is assumed to have been selected for the analog clock. The analog clock frequency is also chosen as hardware option at purchase time. More info in the *Keysight PXIe Chassis M9046A User Manual*, available at Keysight PXI chassis.

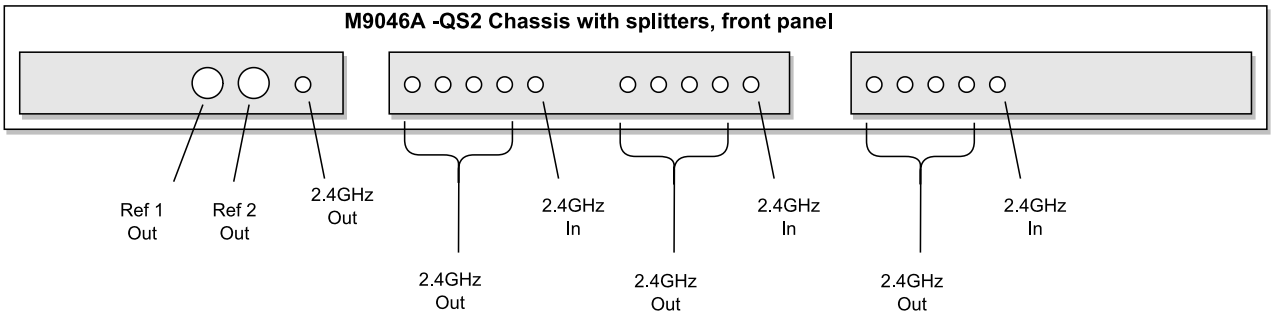
6.2.1. M9046A -QS0 Chassis with no HPRCS and no Analog clock splitters

The following diagram shows the front panel of an M9046A -QS0 chassis.



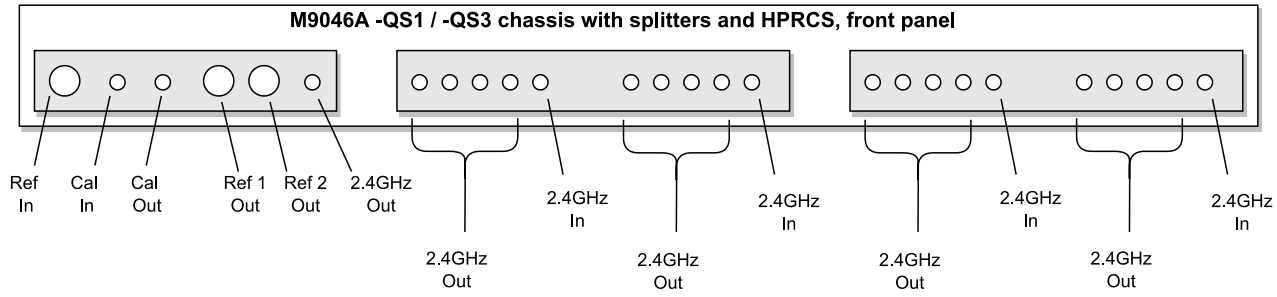
6.2.2. M9046A -QS2 Chassis with no HPRCS, with Analog clock splitters

The following diagram shows the front panel of a M9046A chassis with -QS2 option including the front panel analog clock splitters to ease the distribution of the analog clocks to all modules.



6.2.3. M9046A -QS1/3 Chassis with HPRCS and with Analog clock splitters

The following diagram shows the front panel of an M9046A -QS1/3 chassis with analog clock splitters and Ref In, Cal In and Cal Out for the M9546A HPRCS.



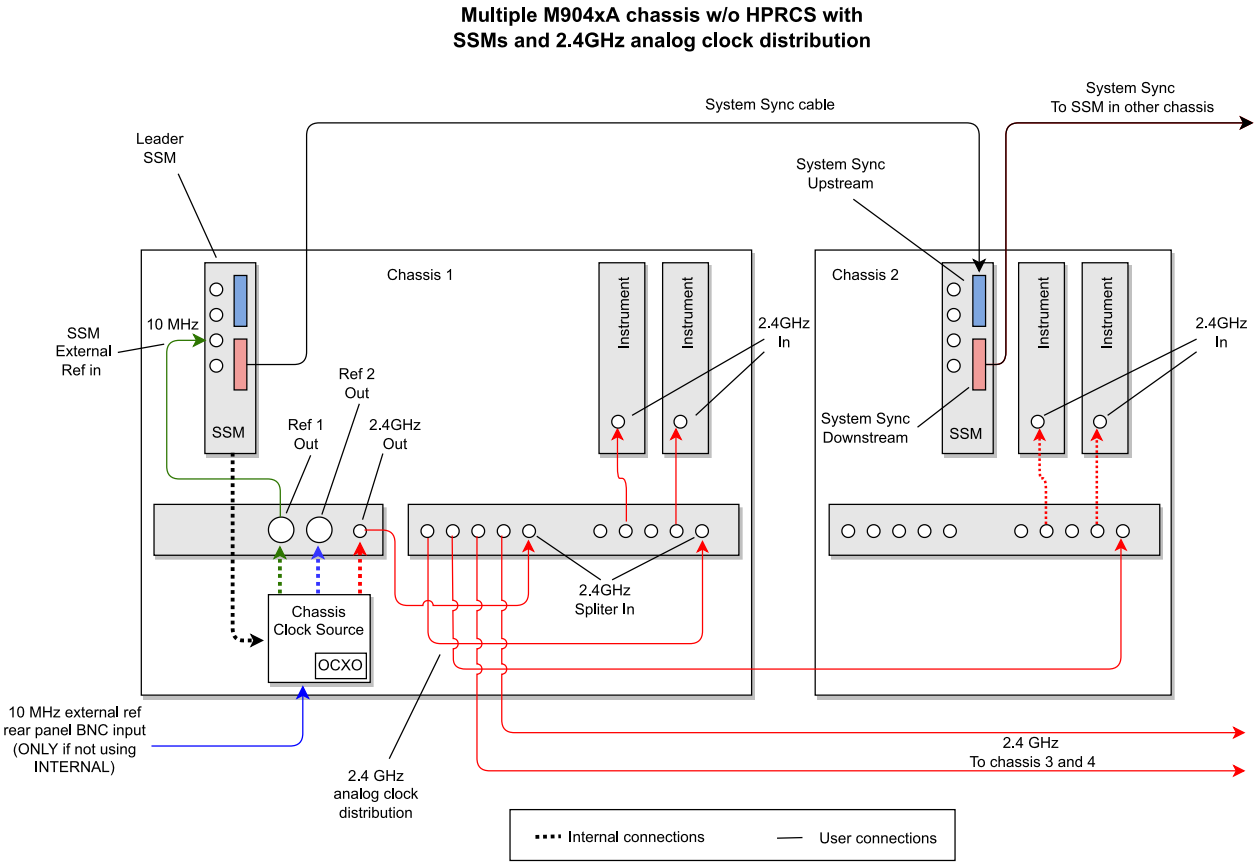
6.3. Single/multiple chassis (M904xA w/o HPRCS) with External Analog clocks and SSMs (Scheme C)

This configuration is for single or multiple chassis with SSMs. This configuration is compatible with PXIe Chassis model M904xA with hardware options -QS0 and -QS2. The internal chassis clock is used as the clock source and this configuration must be defined in the TSE API. The chassis clock must be taken out from the **Ref 1 Out** port on the PXIe M9046 Chassis front panel and must be connected to the **SClk / Ref** In port of the PXIe SSM (see diagram below).

You can use the chassis as the reference clock source with its reference clock set to:

1. **Internal**: Use the chassis internal OCXO.
2. **External**: Using an external reference clock connected to the chassis rear panel 10MHz Ref BNC input.

The following diagram depicts the SSM using the chassis clock (indicated in red) as the clock source. The chassis external reference is indicated by the dotted blue arrow:



NOTE The user is expected to set up all the connections between instruments. This include:

- System Sync cabling between SSMs
- Chassis' Front Panel Ref1 Out to the SSM's Front Panel REF_IN (Only in chassis 1)
- Chassis' Front Panel 2.4Ghz out to splitters and from splitters to instruments
- External Clock Reference to Chassis' Rear Panel

6.3.1. Configuring the M904xA as the system and analog clock source

To use the internal chassis clock, you must:

- Connect the Chassis **Ref 1 Out** output to the SSM's **SClk / Ref In** located in the same chassis.
- In the TSE API you must instruct the SSM to use the chassis clock.

By default, and if it is not specified otherwise, the chassis clock circuitry uses its internal OCXO as the reference clock.

The following code shows how to configure a pair of chassis with SSMs using the chassis clock as the reference clock, options is set to an empty string:

```
# Create SystemDefinition object
ktHvi.SystemDefinition definition("Name")
#
# You must add all necessary follower SSMs depending on the number of chassis
syncModuleLeader = definition.interconnects.add_sync_module(SSM_1, options)
syncModuleFollower = definition.interconnects.add_sync_module(SSM_2, options)
#
# Add chassis
chassis1 = definition.add_chassis(1)
definition.add_chassis(2)
#
# Get the chassis clock
clock_source = chassis1.clock_source
#
# Set as reference
definition.clocking.reference_source = clockSource
#
# Enable the chassis analog clock
clock_output_2_4GHz = ref_chassis.clock_outputs["FP2.4GHzOut"]
clock_output_2_4GHz.set_enabled(True)
```

6.3.2. Configuring the M9046A to use the external reference clock

To use the chassis clock with an external reference clock, you must:

- Connect the external reference clock to the Chassis rear panel's 10 MHz Ref BNC input.
- Connect the Chassis Ref 1 Out to the SSM SClk / Ref In of the SSM in the same chassis.
- In the TSE API you must instruct the chassis to use the external reference clock and set the frequency in Hz.

The following code shows how to set the external reference:

```
# Set the reference mode to use an external reference
# Set clock mode to EXTERNAL and set frequency to 10MHz
clockSource.set_mode(keysight_tse.ClockingReferenceMode.EXTERNAL, 10e6)
```

6.4. Single/multiple chassis with HPRCS, External Analog clocks and SSMs (Scheme D)

This configuration is for single or multiple chassis with SSMs. For this clocking scheme to be used, the first SSM must be in a Keysight M9046A chassis containing an M9546A *High Performance Reference Clock Source* (HPRCS).

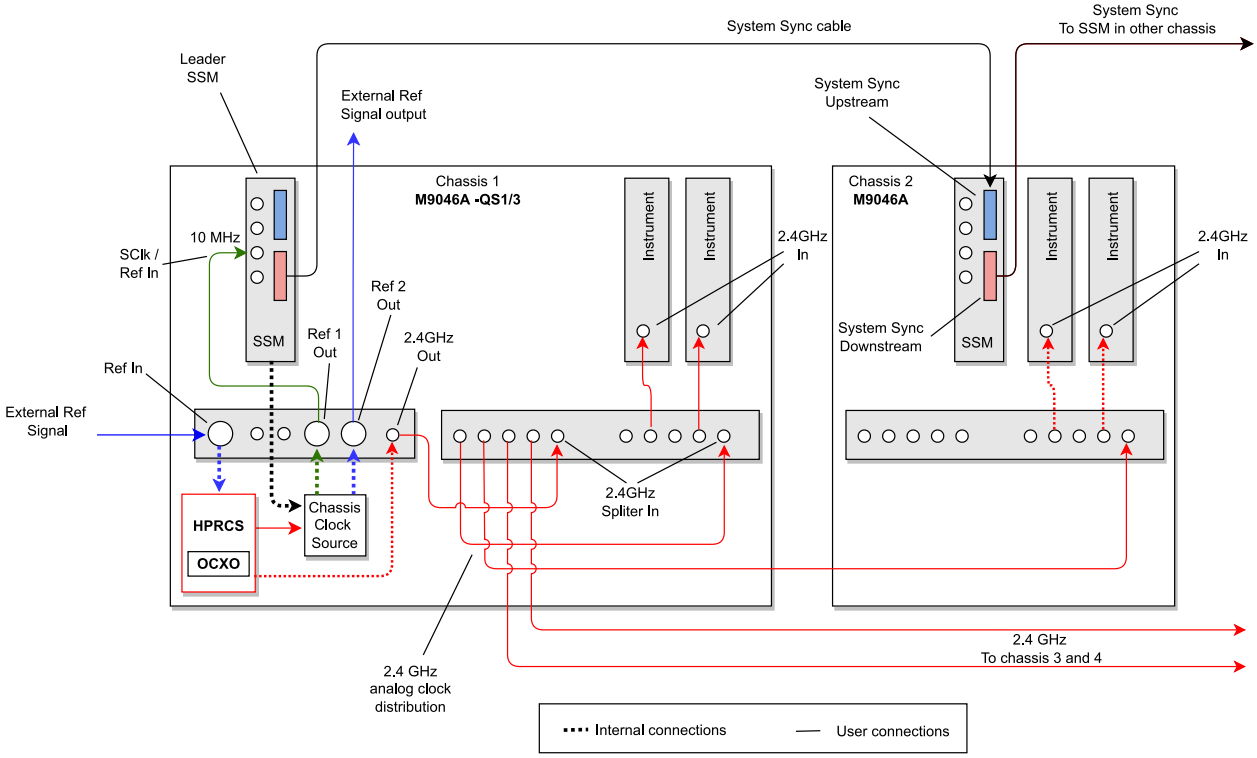
The HPRCS is used as the clock source and this configuration must be specified in the TSE API.

We can use the HPRCS as a clock source with its reference clock set to:

- Internal: Use the HPRCS internal OCXO.
- External: Use an external reference clock connected to the chassis front panel Ref In input.

The following diagram shows the leader SSM using the M9546A HPRCS (indicated in red) as the clock source in chassis 1 M9046A -QS1/3. The HPRCS external reference is indicated by the blue arrow. The distribution of the 2.4 GHz analog clock to up to 4 chassis is also shown. The connection topology and cables used are critical to achieving the optimal channel skew drift performance. For information about how to connect the analog clock to more than 4 chassis, see *Keysight PXIe Chassis M9046A User Manual* available at Keysight PXI chassis.

Multi-chassis with M9046A chassis with M9546A HPRCS as clock source for external 2.4GHz analog clock distribution and SSMs



NOTE The user is expected to set up all the connections between instruments. This include:

- System Sync cabling between SSMs
- Chassis' Front Panel Ref1 Out to the SSM's Front Panel REF_IN (Only in chassis 1)
- Chassis' Front Panel 2.4Ghz out to splitters and from splitters to instruments
- External Clock Reference to Chassis' Front Panel Ref In

6.4.1. Configuring the M9046A + HPRCS as the system and analog clock source

To use the HPRCS as clock source, you must:

- Connect the chassis **Ref 1 Out** output to the **SClk / Ref In** of the SSM located in this chassis
- In the TSE API you must:
  - Add the M9046A -QS1/3 chassis with HPRCS to the SystemDefinition.
  - Set the HPRCS to be the clock source.

When no external reference clock for the HPRCS is specified, its internal OCXO is used.

The following code shows how to configure a pair of chassis with SSMs using the HPRCS as clock source, options is set to an empty string:

```
# Create SystemDefinition object
my_system = kthvi.SystemDefinition("MySystem")
#
# Define all necessary SSMs depending on the number of chassis
my_system.interconnects.add_sync_module(SSM_1, options)
my_system.interconnects.add_sync_module(SSM_2, options)
#
# Define chassis
hprcs_chassis = my_system.add_chassis(1)
my_system.add_chassis(2)
#
# Create HPRCS object
clockSource = hprcs_chassis.high_performance_clock_source
#
# Set the HPRCS as the reference clock
my_system.clocking.reference_source = clockSource
#
# Enable the chassis analog clock
clock_output_2_4GHz = ref_chassis.clock_outputs["FP2.4GHzOut"]
clock_output_2_4GHz.set_enabled(True)
```

6.4.2. Configuring the M9046A + HPRCS to use an external reference clock

To use the HPRCS with an external reference clock, you must:

- Connect the external reference clock to the chassis' front panel **Ref In** input.
- Connect the chassis' front panel **Ref 1 Out** output to the **SClk / Ref In** input of the SSM located in this chassis.
- In the TSE API you must:
  - Add the M9046A -QS1/3 chassis to the SystemDefinition.
  - Set the HPRCS to be the clock source.
  - Instruct the HPRCS to use an external reference clock and the desired frequency in Hz.

To use the external reference, set the reference clock mode in the previous code snippet to (defaults to internal):

```
# Set the reference clock mode. Set the HPRCS to use an external reference @10Mhz
clockSource.set_mode(keysight_tse.ClockingReferenceMode.EXTERNAL, 10e6 )
```

6.5. Enabling chassis analog clock output

If you are using an analog clock output from a chassis you must enable it.

The following code shows how to enable a 2.4GHz analog clock output from an M9046A chassis.

```
clock_output_2_4GHz = ref_chassis.clock_outputs["FP2.4GHzOut"]
clock_output_2_4GHz.set_enabled(True)
```

6.6. Enabling the External Analog Clock Source for Instruments

For instruments that require an analog clock, you must set the source and frequency of the analog clock in your SystemDefinition.

You can set parameters for the analog clock:

- The source as internal or external.
- The frequencies of the sources, in Hz.

For external sources, the source selected depends on the analog clock frequencies that the instrument supports.

- If you indicate multiple frequencies, the first external frequency supported by the instrument is selected.
- If none of the external frequencies are supported, and the instrument has an internal clock, the internal clock is selected.
- If none of the external frequencies are supported, and the instrument does not have an internal clock, an error is generated.

The code is:

```
my_system.clocking.enable_external_analog_clocks(frequencies)
```

For example, if you are using a M904xA chassis with the 2.4GHz analog clock reference, add the following line:

```
my_system.clocking.enable_external_analog_clocks([2400e6])
```

Instruments that support an external analog clock are set to use this clock. Instruments that do not support this external frequency are set to use an internal clock. If the instrument does not support the frequency and does not have an internal clock, an error is generated.

6.7. Analog clock distribution guidelines

For your system to work correctly, you must ensure the clock reference distribution is correct. This is especially important for *Radio Frequency* (RF) analog (2.4GHz and above) clock distribution. For the best performance, that is, the lowest skew and drift, follow these guidelines:

- Cable distribution must be symmetrical.
- The number of distribution and amplifier hops to each end point must be the same.
- Cables must be of the same type and same length.
- The temperature of the cabling, distribution and amplifiers hops must be kept as close as possible

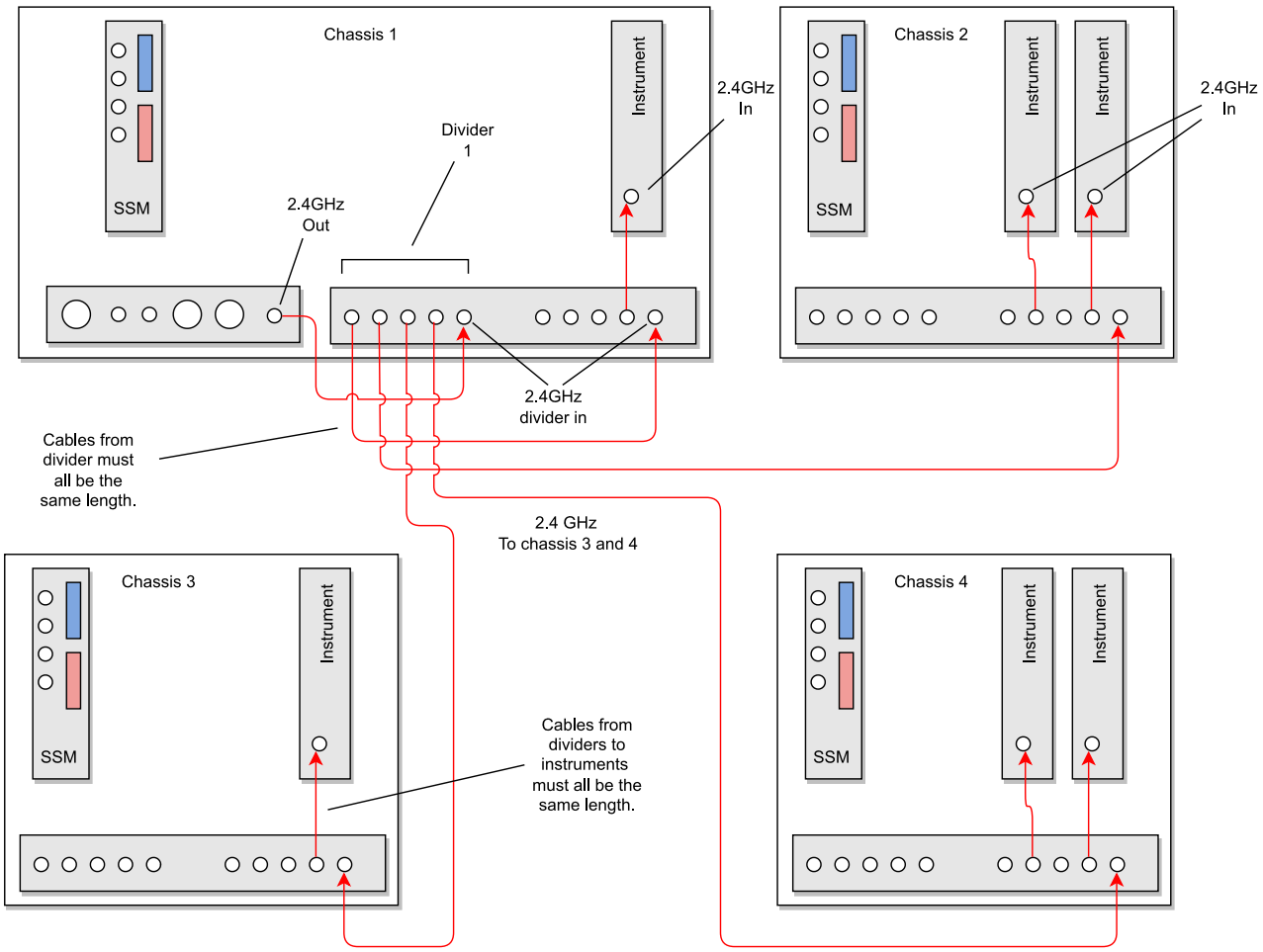
These guidelines ensure that any variances in clock signals are minimized as they travel to individual instruments.

The Keysight M904xA chassis includes as an option 4:1 amplified power dividers built into the chassis to support the balanced-star distribution of external Analog clocks with best performance. These power dividers or splitters are designed specifically for minimizing phase noise, temperature drift, and to maintain the Analog clocks amplitude as it is divided many times. Substituting other power dividers to distribute the Analog clocks may degrade jitter and drift performance, so this is not recommended. In addition the Keysight MCX cables are made of a special material that minimizes their propagation delay change with temperature. Matching the total propagation delay from their common clock source to each instrument causes the propagation delay drifts of the clocks to cancel out between instruments. Note that small spurious oscillations can occur within the amplified power divider when any of the outputs are loaded with certain reflective loads. For this reason, terminating unused outputs with 50 Ohms loads is recommended. It is only necessary to terminate unused outputs of power dividers that are currently being used to distribute the Analog clocks. For more information, see your Chassis and Instrument documentation.

The following diagram shows a multi-chassis setup with M9046A chassis including balanced-star 2.4 GHz analog clock distribution:

The clock originates in chassis 1. It is sent to divider 1 which divides the signal into 4. These 4 signals each go to dividers in all 4 chassis and from there to the individual instruments. Chassis 1 and 3 only have 1 instrument so the divider stage may appear unnecessary, however this would mean the signals to the instruments using less hops than in chassis 2 and 4, resulting in some instruments getting signals before others and the instruments would therefore be out of sync.

2.4GHz analog clock distribution in multi-chassis system.



6.8. Notes on the selection of the best analog clock source for instruments

While it is often convenient for instruments to use their own internally generated analog clocks, the best jitter and drift performance is achieved by using a single common analog clock source generated within the Leader chassis (with or without the HPRCS) and distributing it using the chassis amplified power splitters in a balanced star configuration. This ensures that any low-frequency jitter skew drift is common across the system, minimizing the inter-channel jitter and drift.

In some cases with high channel count configurations, there may not be enough individual copies of the the Analog Clock available from a full balanced star distribution to connect to every instrument. In those cases, a single daisy-chain connection of the Analog Clock between instrument pairs can be used. Noting that the downstream instrument of the daisy-chained pair will have slightly higher skew drift than the non-daisy-chained instrument. Daisy-chained instruments shall have slightly higher skew drift, so these instruments should be the ones in the system which have the lowest bandwidth. For example, in systems which employ the M5201A Downconverter and the M5200A Digitizer, which are typically used in pairs, it is best practice to route the Analog Clock to the downconverter first and then daisy-chain the downconverter's Analog Clock output to the digitizer's Analog Clock input. This is because the 2 GHz digitizer is less sensitive to the same amount of channel skew than the 16 GHz downconverter.

7. System Initialization

The TSE API enables you to control the process of system initialization and clock alignment.

The following describes the steps you take to initialize the system for a number of different scenarios. For more information about initialization options available in the TSE API, see the section *System Initialization* in the *PathWave Test Sync Executive User Manual* available here.

The system must be initialized after every power cycle, a change in the system hardware configuration, or a change in the clock configuration. The initialization procedure you need to perform depends on:

- The instruments in the system.
- Your required channel skew accuracy.
- If the hardware or clock configuration has changed.
- If a system initialization has been performed.

7.1. Example of System Initialization and Operation

To use the TSE API to initialize and run real-time operation in your system, there are two main procedures that you must follow:

1. System Warm-up and Calibration
2. Normal Operation

There are also a number of use cases that are variations on these main procedures. The following text describes these procedures along with the use case variations.

NOTE The initialization process requires access and control of all of the hardware resources, so it is important that these resources are not already in use by another application or HVI instance already loaded to hardware. An exception is thrown if any of the hardware resources are already in use.

7.1.1. System Warm-up and Calibration

The system warm-up must be performed every time the system is turned on or the hardware configuration is changed. This is to enable all of the components to reach a stable and repeatable operating temperature. Once the system is warmed-up, the system can be initialized using the stored System Calibration data.

The System Calibration must performed in these cases:

1. The very first time that the system is put together and powered-on.
2. When relevant hardware changes are made that require a new system calibration. These hardware changes include:
   a. Adding/removing a chassis in your SystemDefinition object.
   b. Adding/removing any instrument that requires clock alignment calibration data, such as an M5300A module or M5201, or changes the operating temperature of the system.
   c. Changing the cable connections between System Synchronization Modules, even replacing a cable with a similar one with a different serial number.
   d. Changing any of the external System Clock or Analog Clock cable connections, even replacing a cable with a similar one with a different serial number.
   e. Making any change to the clock configuration, even if it is only from the TSE API. This is because this triggers the usage of different clock sources or signal paths.
3. Other situations where the system calibration should be updated.
4. On rare occasions, a component in the system can move into an invalid state and a reset of the calibration might be required. For more information, see *System Troubleshooting* in the ***System Setup Guide*** .

NOTE Warning : Resetting the system calibration shall in turn require you to recalculate the User Calibration for some instruments. Observe extreme caution when doing this to avoid costly time-consuming recalibration.

Procedure steps:

1. Power-on the system
   a. Power-on all of the chassis. After this is complete, if you are using an external chassis controller, power it on.

2. Connect to all the instruments
   a. For example: `instrument = ktm5300.KtM5300x(resource_id, query, reset, options)`

3. Create a SystemDefinition using the TSE API and the instrument drivers:
   a. Create a SystemDefinition object that we refer to here as *my_system*. Use the *my_system* object to define all the hardware resources in your system: chassis, SSMs, instruments, clocking configuration, reference clock source, etc.
   For example: `my_system.chassis.add(1)`, `my_system.clocking.reference_source = chassis.clock_source`
   b. Add the HVI Engines of each instrument to the SystemDefinition object.
   For example: `my_system.engines.add(instrument.hvi.engines.main_engine, "MyEngine")`

4. System Initialization for Warm-Up
   a. Execute `my_system .initialize(keysight_tse.AlignmentModes.FULL | keysight_tse.AlignmentModes.PRE_CALIBRATION)`. The `PRE_CALIBRATION` flag indicates there is no need to apply any previously stored system calibration values because the system is warming-up. This enables the system to execute code without calibration related errors. After this step, instruments may present channel skew errors which are compensated by the next steps.

5. Wait for System Warm-Up
   a. Wait for the required warm-up time, this can range from a few minutes to about 30 minutes. The actual time typically depends on the type and number of instruments in the system, clocking configuration, etc.
   b. For detailed warm-up time information, see your instrument documentation, for example: *M5300 RF AWG User's Manual* .

6. System initialization to perform System Calibration
   a. Using the SystemDefintion created in step 3, run `my_system.initialize(keysight_tse.AlignmentModes.FULL | keysight_tse.AlignmentModes.RESET_CALIBRATION)` to generate internal system calibration data. At first system turn-on, no previous calibration data is expected to be available.

7. Calculate User Calibration or channel deskew (Optional)
   a. This operation is optional and consists of correcting analog channel skews introduced by cable and signal path delays. Note that in some instruments, the User Calibration must be re-calculated when a System Calibration is executed. For information about how to do this, see your instrument documentation.

8. Ready for Normal Operation

Use Cases

Use Case Scenario	Description
First system start-up and calibration	The very first time that the system is put together and powered-on, you must execute a full warm-up and calibration procedure to achieve the best system performance and repeatability: • Execute all steps #1 to #7 above.
System start-up using existing calibration	If the system has already been calibrated for the current hardware configuration, then, to reuse the existing calibration to configure the system, wait for the system temperature to stabilize then apply the existing calibration: • Execute steps #1 to #5 above. • Skip steps #6 and #7 *System initialization to perform System Calibration* and *Calculate user calibration or channel deskew* , and run `my_system.initialize (keysight_tse.AlignmentModes.FULL)`.
Simplified uncalibrated system start-up	If you want to use the system for test development, or you can tolerate analog channel drift of up to 50ps across reboots/power-cycles: • Execute steps #1 to #4 above. • Skip steps #5 to #7 *Wait for System Warm-Up* , *System initialization to perform System Calibration* and *Calculate user calibration or channel deskew* .

NOTE **System hot boot-up:** If the system is already warmed-up to the calibration operating conditions, for example after a system restart, you can skip the steps #4 and #5 *System Initialization for Warm Up* and *Wait for System Warm-Up* .

7.1.2. Normal Operation

Once the system is warmed-up and the system calibration has been done, users can use the the TSE API to execute real-time operations:

NOTE Note that if it is the first system start-up or you have introduced any of the HW changes that require new System/User Calibration you must execute the *First system start-up and calibration* use case described in the *System Warm-up and Calibration* procedure.

Procedure steps:

1. Connect to all the instruments, if not already connected.
   a. For example: `instrument = ktm5300.KtM5300x(resource_id, query, reset, options)`
2. Apply user calibration to instruments, You only need to do this if it is required, the user calibration data is available, and it has not been applied already.
   a. The user calibration is calculated during the *System Warm-up and Calibration* process. For information about how to apply existing calibration, see your instrument documentation, for example: *M5300 RF AWG User's Manual* .
3. Create a SystemDefinition object, or reuse an existing one.
4. Initialize the SystemDefinition object (Optional)
   a. Run `my_system.initialize()`. This call executes the minimal or default initialization, provided a Full Initialization has been executed already as described in the *System Warm-up and Calibration* procedure. If the full initialization has not been executed, this step requires calibration data. If the calibration data is not available this operation will fail. To run the system initialization without calibration you can specify the `PRE_CALIBRATION` flag: `my_system.initialize (keysight_tse.AlignmentModes.PRE_CALIBRATION)`
   b. Note that you can skip the call to `my_system.i nitialize()` because the minimal or default initialization happens implicitly in steps #5 and #7 described below.
5. Create a Sequencer object
   a. For example: `sequencer = keysight_tse.Sequencer("MySequencer", my_system)`
   b. Note that the sequencer creation operation implicitly executes a default initialization, this is equivalent to calling `SystemDefinition:Initialize()`.
6. Create an HVI object
   a. For example: `hvi = sequencer.compile()`
   b. The `Hvi` object is created by compiling the Sequencer object after all the HVI Sequences have been programmed.

7. Load HVI to HW
   a. For example: `hvi.load_to_hw()`
   b. Note that the load_to_hw() operation implicitly executes a default initialization, this is equivalent to calling `SystemDefinition:Initialize()`.
8. Run HVI
   a. For example: `hvi.run( hvi.no_timeout)`
9. Release HW
   a. For example: `hvi.release_hw()`

NOTE **Forcing a full initialization.** You can optionally force a full initialization. Forcing the full initialization can be useful to unblock a system if it is in a bad state, when some temporary hardware changes in the system are done such as reconnecting cabling using the same cables, or in general when it is useful to ensure the system is fully initialized to discard any previous state. To force the full initialization run:

1. `my_system.initialize(keysight_tse.AlignmentModes.FULL)`.
2. Or if you are using the system without calibration, add the `PRE_CALIBRATION` flag: `my_system.initialize(keysight_tse.AlignmentModes.FULL | keysight_tse.AlignmentModes.PRE_CALIBRATION)`

NOTE **User Calibration not required or already applied:** If user calibration is not required or has already been applied to the instruments, you can skip step #2 ***Apply user calibration to instruments.*** For more information on how to handle User Calibration in instruments, see your instrument documentation.

8. System initialization with TSE Service and Multi-Host support

TSE Service and KDI offer capabilities to configure and initialize single and multi-host systems. TSE Service works without KDI, but it is recommended to use it together with KDI for multi-host applications or to benefit from the possibility to share instruments across applications, in particular, delegate in TSE Service the boot-up initialization of instruments, to speed up application execution later.

8.1. TSE Service Overview

TSE Service offers extended capabilities for system initialization and configuration:

1. Extend the use of TSE to Multi-Host architecture (see *Free-Running mode*). If you want a system with more than 6 PXIe chassis or remote connectivity, then you require a Multi-Host system.
2. Enables the user to define a system configuration using one or more `.yml` files to automate the complete system initialization (a time-consuming task), and execute it at host boot-time (see *Leader-Follower mode*). This can be used with both single and multi-host systems.

8.1.1. TSE Service Running Modes

TSE Service can be configured to operate in one of 2 different modes:

Free running Mode:

> This mode is mainly intended for Multi-Host systems because it enables a client application to access resources distributed across multiple hosts. The client application is responsible for defining the topology of the system using the SystemDefinition class to add chassis, SSMs, HVI engines, etc, and to run the system initialization.

Leader-Follower Mode:

> In this mode TSE Service in the leader host automates the complete system initialization. This initialization is executed at boot time, speeding up the applications execution later. The system is defined in the leader in a `system_definition.yml` configuration file. The client application does not need to define or initialize the system, it just creates the application SystemDefinition that connects to the leader TSE Service and gets all the system information automatically.

8.2. KDI Overview

Keysight Distributed Infrastructure (KDI) provides network infrastructure services and simplifies deployment of a distributed environment.

KDI provides:

- Management of instruments drivers (opening/close) and discovery across the network.
- Instrument remote access.
- Authentication service (KDIS) for secure access.

With KDI, you can use KDI Resource IDs to open or access instruments across the network, see Resource IDs for Accessing Remote Resources section.

TSE Service makes use of KDI internally to open and initialize instruments at boot-up and get them ready for use by user applications later, see *TSE Service Free-Running Mode* section.

8.2.1. KDI Authentication Service (KDIS)

KDIS is a standalone, cross-platform, and lightweight gateway service that authenticates connections. The authentication service enables secure communication within the KDI Fabric.

It includes the KDIS Admin UI that enables you to:

- Add, delete, and update users.
- Register devices (test stations and instruments).
- Register processes with credentials.

On receiving an authentication request, KDIS has the capability of performing the following actions:

- Validate the credentials against the database.
- Validate previously issued tokens and authorize a connection.
- Issue a public key for more efficient validation.

NOTE Keysight recommends that only one KDIS instance is installed in a system. In this case, any host installing a KDI Client automatically detects the KDIS service for authentication. If for some reason more that one KDIS must be installed, then it is the responsibility of the user to direct each KDI client to the correct KDIS host. See the KDI documentation for more information.

8.2.2. KDI Clients

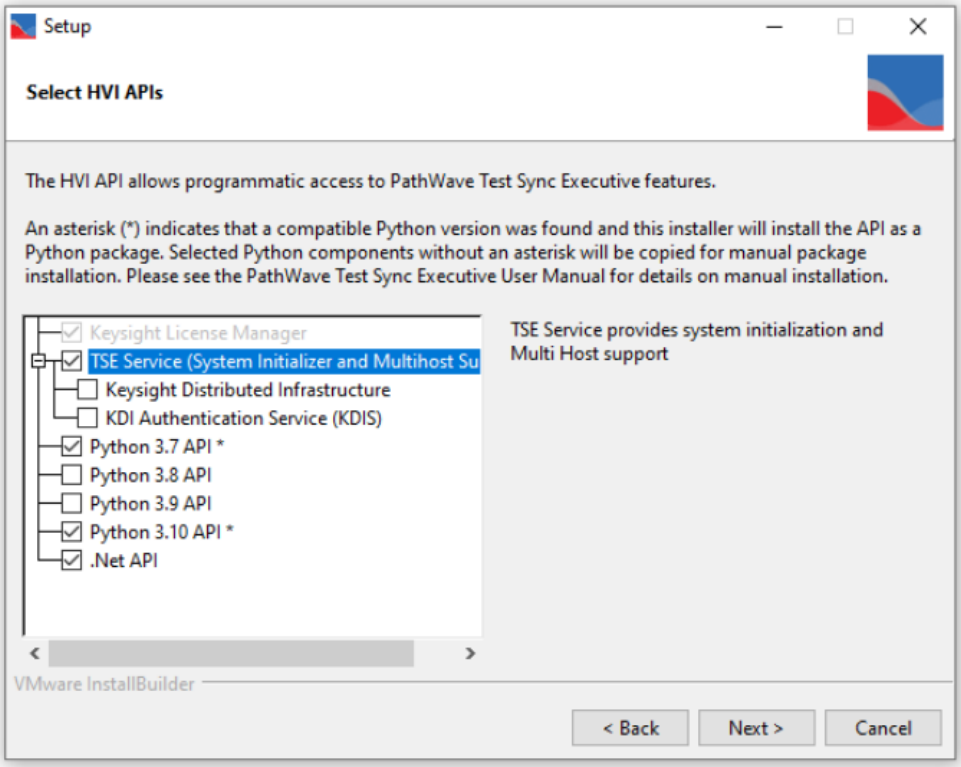
The KDI Client is a service that must run in any host that needs access to the KDI fabric and services such as: securely access information, status, and control features for nodes in the KDI Fabric.

8.3. TSE Service and KDI installation

Both TSE Service and KDI infrastructure can be installed from the TSE Installer with the appropriate selections.

8.3.1. TSE Service installation

To install TSE Service, select the check box labelled TSE Service in the installer. TSE Service is typically installed with KDI, details of the KDI installation options are shown in the section *KDI Installation Overview.*



8.3.2. KDI Installation

KDI can be installed as part of PathWave Test Sync Executive installation when enabling TSE Service.

When Keysight Distributed Infrastructure is selected, TSE Service is configured to be started automatically at windows boot-up. The default configuration after installation will start TSE Service in Free-Running mode, configured to autodetect and open all PXI chassis and Instruments supported by TSE Service.

Once TSE Service is selected, to enable KDI installation, you can select:

Keysight Distributed Infrastructure (KDI client)

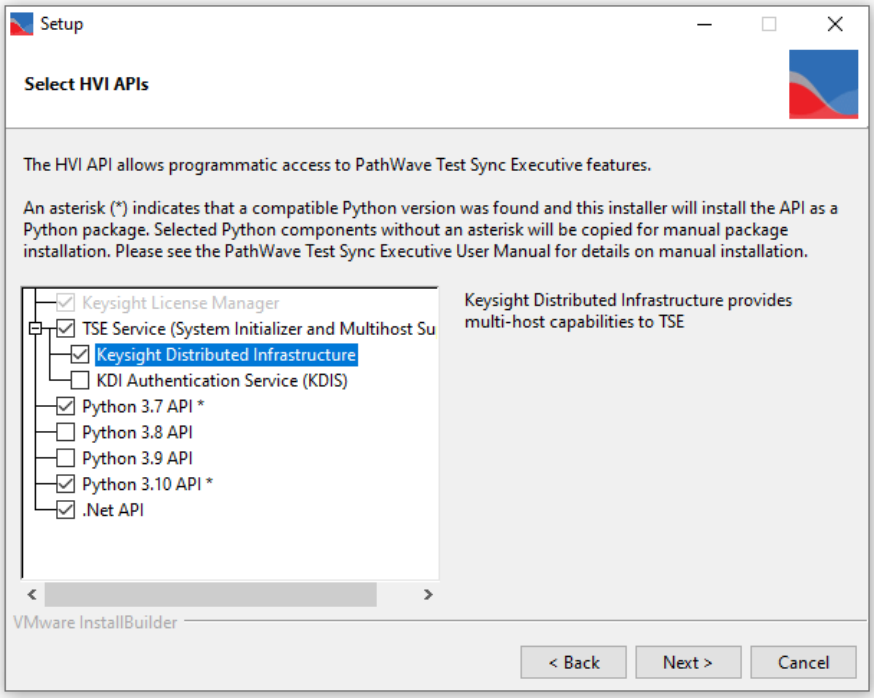
KDI must be installed on every host.

KDI Authentication Service (KDIS)

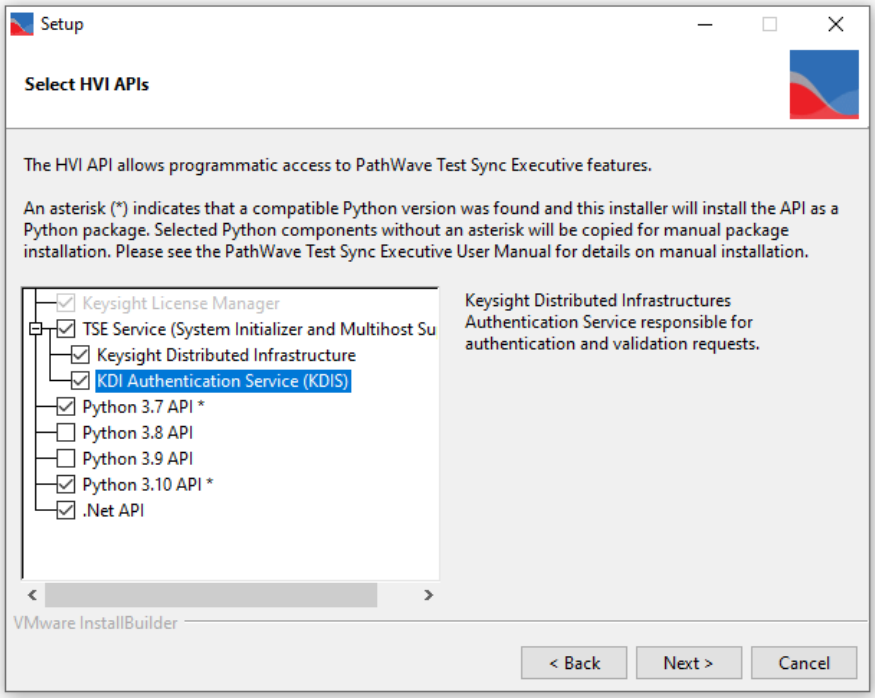
This is installed on only one host.

NOTE Because of a limitation in KDI infrastructure (up to at least release 3.0.187), if you intend to use the TSE Service in Leader-follower mode, then the KDIS must be installed in the Leader TSE Service host.

To install the KDI client, select **Keysight Distributed Infrastructure**. Do this for every host.



To install the KDIS and the KDI client, select **Keysight Distributed Infrastructure** and **KDI Authentication Service (KDIS)**. Do this in one host, in a Leader-Follower system, this must be the leader.



NOTE Keysight recommends that only one KDIS instance is installed in a given network. In this case, any host installing KDI Client will automatically detect the KDIS service for authentication. If for some reason, more that one KDIS must be installed, then it is the responsibility of the user to direct each KDI client to the right KDIS host. For more information see the KDI documentation.

For full installation details of PathWave Test Sync Executive, see the *PathWave Test Sync Executive User Manual*.

Once KDI has been installed, you can configure the KDI Authentication Service.

8.3.3. Configure the KDI infrastructure and Authentication Service

KDI infrastructure requires KDI Authentication Service (KDIS) to operate. The host with the KDIS installed is called the root-node. The KDIS instance must be the same for all hosts in a lab or setup that are intended to work together, basically the group of hosts that work together must all have the same root-node. Keysight recommends that KDI service instances in all hosts are configured explicitly to point to the correct root-node or KDIS (see *Configure KDI* section below and for information or troubleshooting check the KDI documentation).

There are a number of steps to configure the KDI Infrastructure:

- *Configure KDI Authentication Service* (KDIS) - the root-node:
  - Configure admin credentials.
  - Create an initial user account.
  - Add one or more Users.
- Configure KDI Clients - this applies also to the root-node.
- Restart the KDI Services.
- Accept KDI Clients in KDIS.

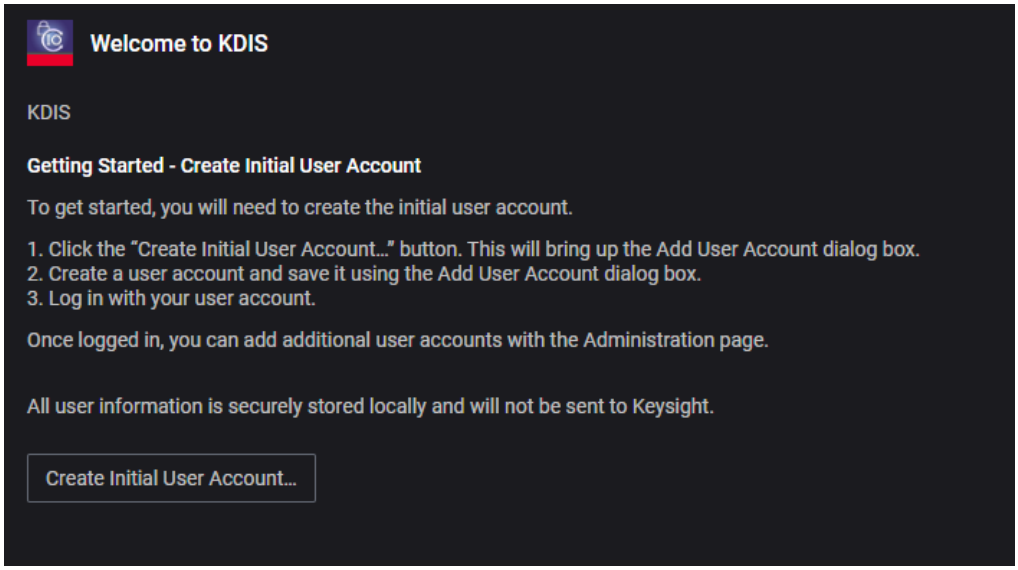
For version kdi version 3.0

8.3.3.1. Configure admin credentials in KDIS:

In the root host (or leader), o pen the KDIS Admin UI at the following URL:

- For KDI versions prior to v3.2 : https://localhost:8886/
- For KDI versions starting from v3.2 : https://localhost:7701/login

The page might give some security warnings, you can ignore these at this stage:



8.3.3.2. Create an initial Admin User Account in KDIS

You must first create an admin account, this enables you to access the KDIS User Interface and administer the system.

Click **Create Initial User Account...**

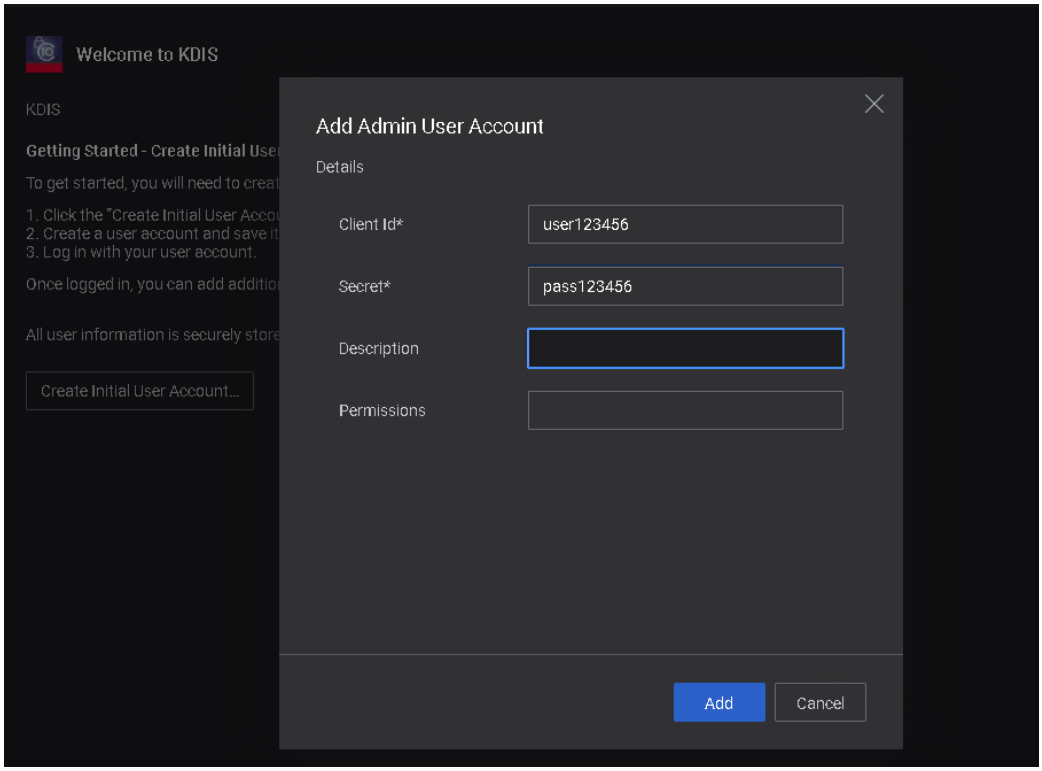
The **Add Admin User Account** page opens.

Add a username (Client Id) and password (Secret)

These are the credentials for the admin account.

The following image shows the **Add Admin User Account** page with example credentials:

- ClientID: user123456
- Secret: pass123456



8.3.3.3. Add additional users in KDIS:

To enable users to co nnect to an instrument with KDI requires a user (also called clients) registered in KDIS with a username (Client Id) and password (Secret).

NOTE The KdiUser and KdiPassword required when opening instruments in the user application refer to the Client Id and Secret registered in KDIS.

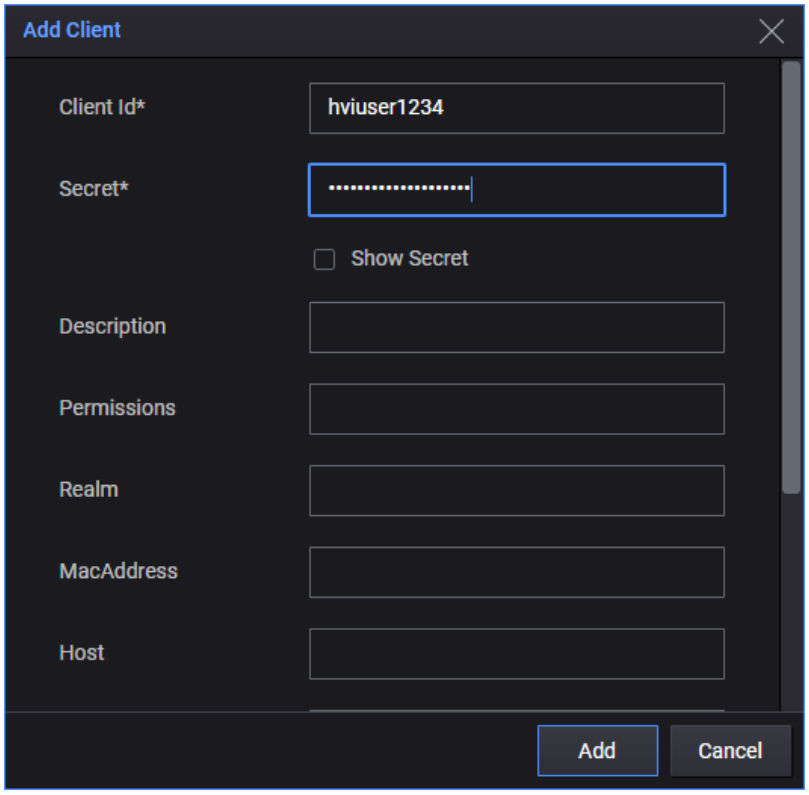
These are also used for accessing the Leader TSE Service in Leader-Follower mode.

For example, to add User "*hviuser1234* " with password "*hviuser1234Password*" click **Add**, and introduce the following information:

- Client Id: *hviuser1234*
- Secret: *hviuser1234Password*

Accept the credentials that KDIS adds.

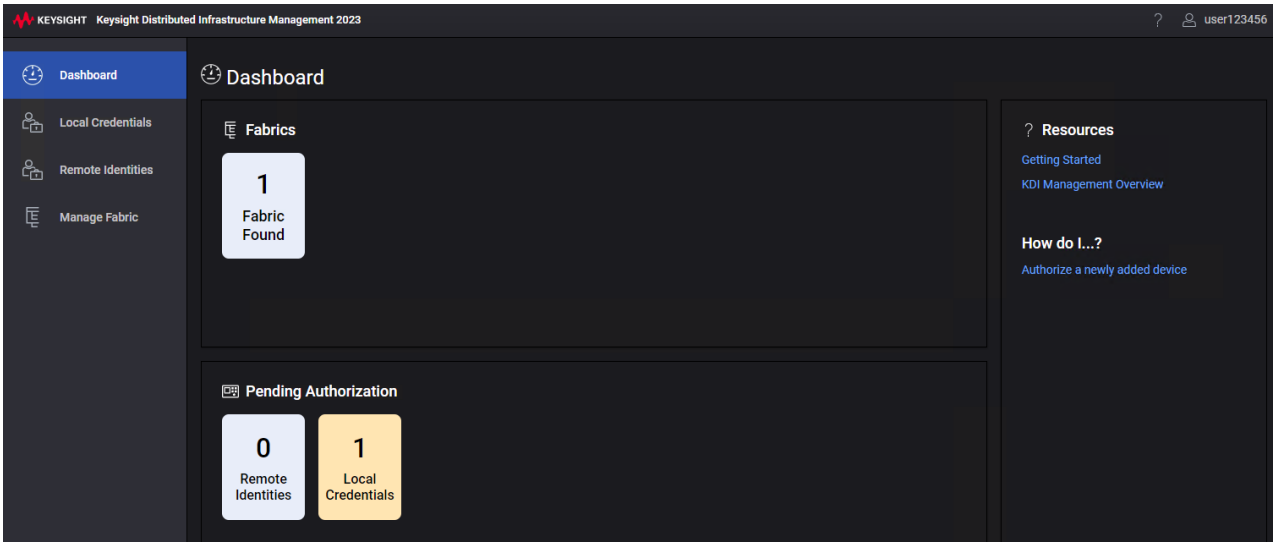
For example:



NOTE After setting up for the first time or updating the KDIS configuration in the root node, the KDI service must be restarted. See Restart the KDI service section below.

8.3.3.4. Add additional users in KDIS 3.2:

In KDIS 3.2 and above requires you to set an additional parameter when you add a user.



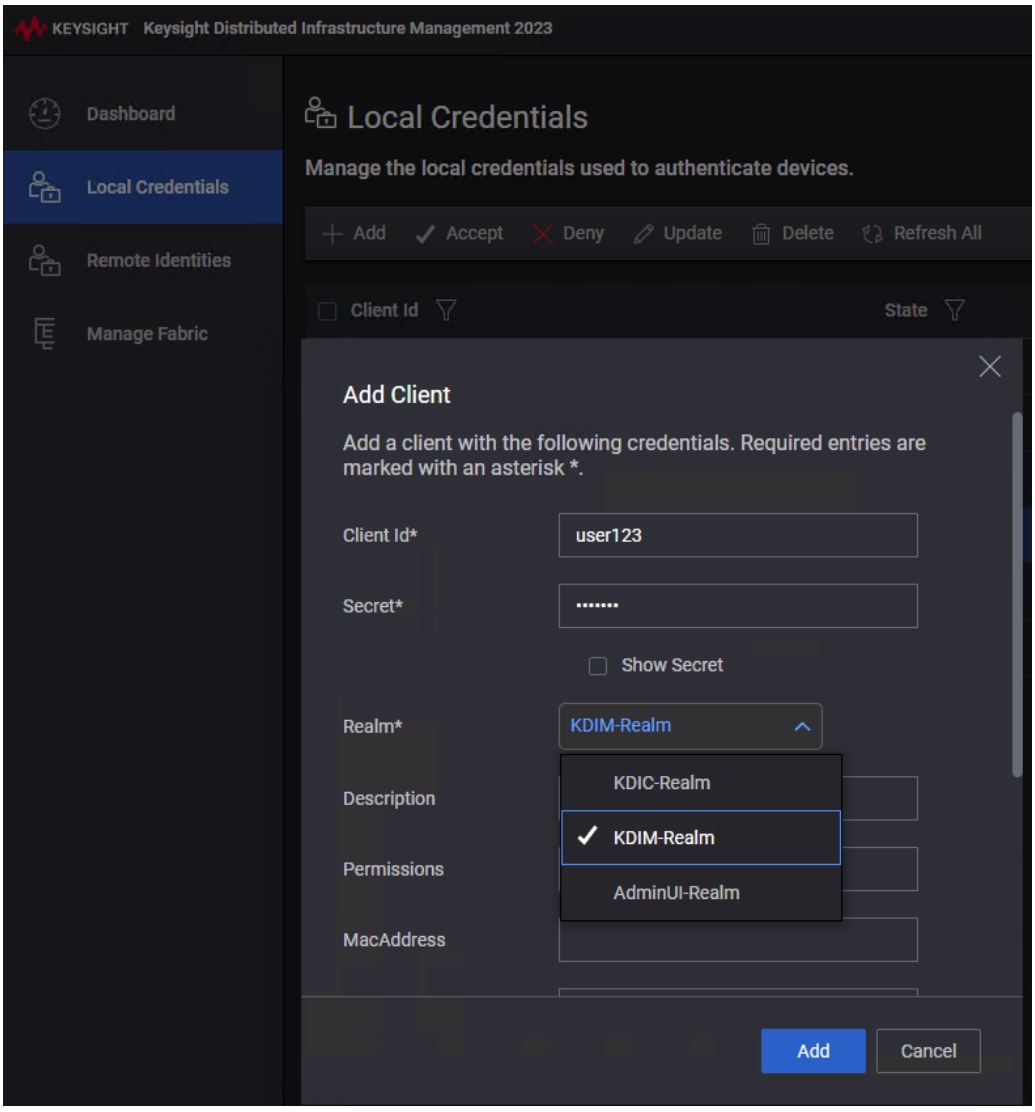
To add additional users in KDIS version 3.2, go to the **Local Credentials**.

You must set the *Realm* parameter when you add a user.

For example, to add User "*hviuser1234* " with password "*hviuser1234Password*" click **Add**, and introduce the following information:

- Client Id: *hviuser1234*
- Secret: *hviuser1234Password*
- Realm: *KDIM-Realm*

and accept these credentials.



8.3.3.5. Configure KDI Clients to find KDIS

Every host in the KDI infrastructure must have a root-node (KDIS host) assigned. It is important for all hosts that must work together to have the same root-node (KDIS) assigned, for instance for TSE Service in Leader-Follower mode.

For all hosts including the root-node:

- Open the **kdi.yml** file located at **C:\ProgramData\Keysight\Distributed Infrastructure**
- Set upstreammanager to the address of the host with KDIS installed, this is the root node or host.
  - For TSE Service to work properly in leader-follower mode, the host configured as TSE Service Leader, must also be configured as root-node.

In all non-root hosts the kdi.yml file must include:

```
...
# Specify KDIS host and port
# This section allows instrument drivers to find KDIS properly (do not use inline comments to avoid issues
with instrument drivers)
upstreammanager:
    host: "myKdisHostname"
    port: "9090"
...
```

For the root-node the configuration can use "localhost":

```
...
# Specify KDIS host and port (do not use inline comments to avoid issues with instrument drivers)
# This section allows instrument drivers to find KDIS properly (do not use inline comments to avoid issues
with instrument drivers)
upstreammanager:
    host: "localhost"
    port: "9090"
...
```

8.3.3.6. Restart the KDI Services

To restart the KDI service in each host, user needs to do one of the following:

- Go to Task Manager → Services → KeysightDistributedInfrastructureService and Right Click →Restart
  or
- Use the following command:

  ```
  C:\Program Files\Keysight\Distributed Infrastructure\DistributedInfrastructureService.exe -service
  restart
  ```

You can check if KDI is running successfully by checking the log, located in:

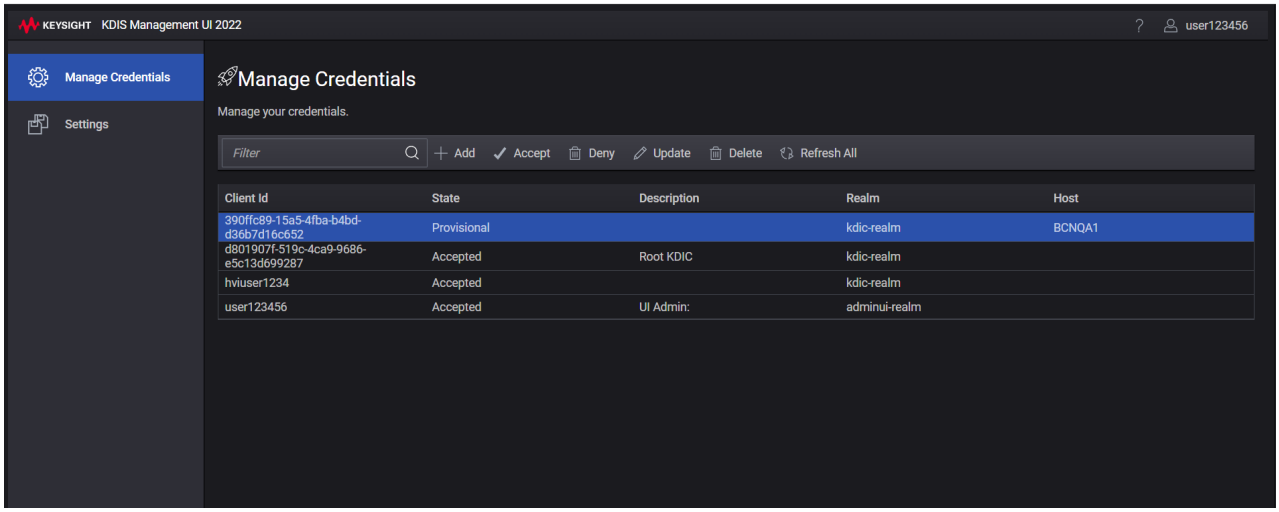
C:\ProgramData\Keysight\Distributed Infrastructure\Logs

8.3.3.7. Accept Clients in KDIS

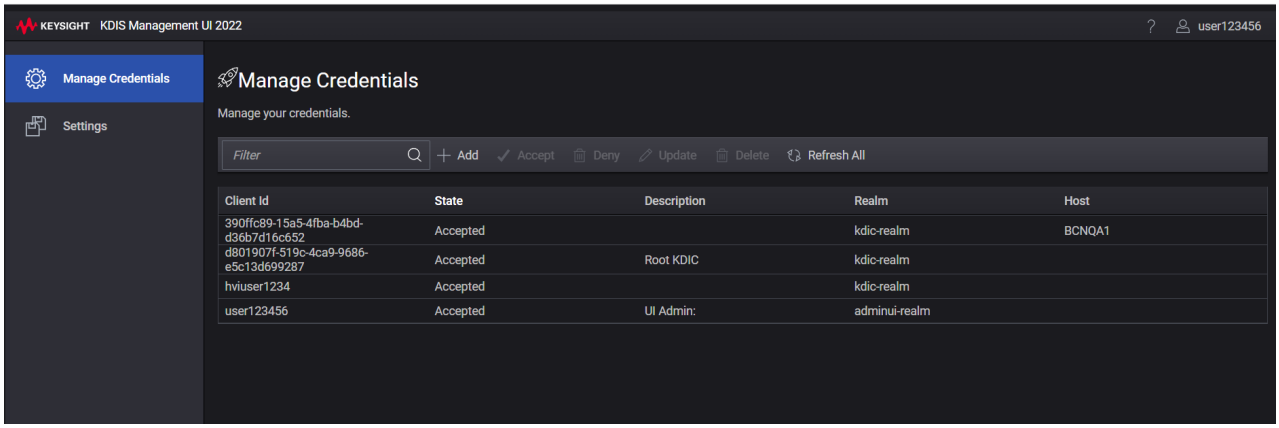
In KDIS, all hosts are referred as clients. Once the KDI instances in the non-root hosts have been properly configured and restarted, they will attempt to connect to the authentication server. On this first connection attempt KDIS will accept the client as "Provisional" and until it is accepted this host will not be able to use the Authentication Services. To Accept a host into KDIS you must:

1. Open the KDIS Admin UI:
   a. For KDI versions prior to v3.2 : https://localhost:8886/
   b. For KDI versions starting from v3.2 : https://localhost:7701/login
2. log in to KDIS
3. Each non-root host (client) with *Provisional* state, accept it if it is a valid host.

The root-node where KDIS is installed also appear in the client list, but it is accepted by default. The example below shows Host BCNQA1 with provisional state:



Once accepted BCNQA1 host will appear with the new state:



8.4. TSE Service execution

TSE Service must be started and have complete its initialization before it can be used by client applications. There are two ways to launch TSE Service:

- Automatic TSE Service Execution at boot-up with KDI (recommended).
- Manual TSE Service Execution (without KDI).

8.4.1. Automatic TSE Service Execution at boot-up with KDI (recommended)

This is the recommended option for most users. KDI launches TSE service automatically at host boot-up. When KDI is selected during the TSE installation, TSE installer automatically registers the TSE Service to be launched by KDI.

In addition to all of the KDI advantages (see About KDI section above), like not needing to specify the TCP port to connect to resources or TSE Service, there are 2 key advantages for using KDI:

- You don't have to specify KDI Users/Passwords in the TSE Configuration files.
- Launching TSE Service at host boot-up without the need for any user to log in, speeds up the system initialization and how soon the system is ready for an application to use it. Application execution is also faster, since chassis, SSMs and instruments are opened at boot time per `tse_config.yml` before the user application runs. If TSE Service is configured in Leader-follower mode, then the complete system initialization is executed automatically at host boot-up time, eliminating this time-consuming process from the application execution.

NOTE When TSE Service runs as a service launched by KDI, it runs as the windows SYSTEM user. It is important that this user is configured properly, for instance licensing, see *TSE Service Leader licensing requirements* in the leader-follower mode section for more details.

8.4.1.1. Restarting TSE Service when launched by KDI

When using KDI, to restart TSE Service, you must first stop the current instance of TSE Service (see *Shutting Down TSE Service* section), and then trigger the local KDI Client to restart the TSE Service (see KDI documentation for details). If TSE Service is not running, it will be started automatically under any of these two conditions:

1. KDI is restarted. See *Restart the KDI Service* section above.
2. When `TseService-2023B.yml` in the `C:\ProgramData\Keysight\Distributed Infrastructure\Services` folder is saved (even without changes).

NOTE When TSE Service is configured to by launched by KDI, do not launch it manually from command line, the KDI authentication required for TSE and KDI resource IDs does not work the same. For manual execution it requires to explicitly specify the KDI User and Password in the `tse_config.yml` , see TSE Service Free-Running mode section for more details.

8.4.2. Manual TSE Service Execution (without KDI)

Starting up TSE Service without KDI requires the user to launch TSE Service explicitly by executing:

- `"C:\Program Files\Keysight\PathWave Test Sync Executive 2023B\TseService\bin\TseService.exe"`

The supported command line options are shown below.

TSE Service startup can also be automated by the user without using KDI:

1. Use the windows Start menu, or a similar capability to launch it automatically at windows log-in.
2. Launch it from a user application or service.

It is important to make sure TSE Service completed initialization before any user application tries to access resources managed by TSE Service. Also, note that in multi-hosts setups the user must make sure TSE Service is launched in all hosts.

Once running, you must use TSE-TCP Resource IDs to access chassis and SSMs resources (see *Resource IDs for Accessing Remote Resources*). For this reason, if you are not using KDI to launch TSE Service, it is recommended that you specify the TSE-Service TCP port explicitly and don't leave it to automatic selection. This means that the port is fixed and can be specified without issues in the TSE-TCP Resource IDs. The TCP port can be specified in the `tse_config.yml` file (see *TSE Service Free-Running Mode section*) or as a command line option (see details below on the supported command lines options).

NOTE If you do not specify the port or leave it as automatic, you can find out the port TSE Service is listening on in the TSE Service log. Note that it is not guaranteed that the same port will be automatically selected on different executions.

It is important to note that when not using KDI to launch TSE Service, when configuring TSE Service to open instruments at boot up, you must also specific the KdiUser and KdiPassword in the `tse_config.yml`.

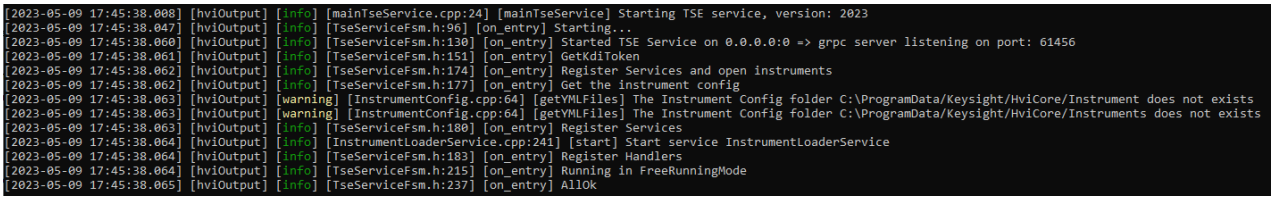
NOTE It is recommended you use KDI to launch TSE Service automatically in those cases where TSE Service is configured to open instruments at start up.

8.4.3. TSE Service Log output

TSE Service outputs information that is sent to the console including:

- Details on all operations executed.
- State of the Service.
- Errors and warnings generated during execution.

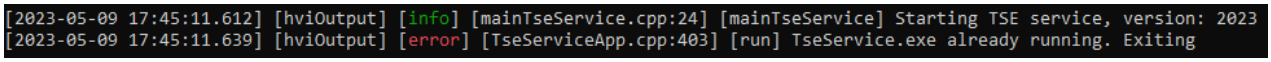
The Log output looks like this:



In the screenshot above, the last line indicates that this instance of the TSE Service has completed the free-running mode initialization successfully and is ready to receive connections from client applications or a leader TSE Service. Messages are identified with three categories:

1. [info] => These provide information about what has been executed, no action is required.
2. [warning] => These are not expected conditions, but not critical to prevent to continue the TSE Service operation. Keysight strongly recommends you to verify any warning to make sure they are not an issue for a given application.
3. [error] => These are critical situations that do not permit TSE Service to continue normal execution. In these situations, TSE Service shall wait some time and then retry, restarting the TSE Service.

As an example of an error condition, If you try to start TSE Service when it is already running, you get this error:



8.4.3.1. Saving TSE Service Log output to a file

By default TSE Log is sent only to the console. It is possible to also save the TSE Service Log output to a file, for that you must add an environment Variable:

Environment Variable	Value	Description
`TSE_SERVICE_LOG_PATH`	Any existing valid path in your system, For example: `c:\\LogsTseService`	Path where the Log output file will be saved

When using KDI to automatically launch TSE Service, console output is not available. With KDI, the environment Variable must be specified in the KDI TSE-Service configuration file located in:

C:\ProgramData\Keysight\Distributed Infrastructure\Services\TseService-2023B.yml

The following snippet shows an example of the configuration file, this is not the default configuration:

```
# NOTE: blank lines are not allowed, they signal the end of the document
#
# name, version, and executable path all match driver as installed.  Do not edit.
name: TseService
version: 2023B
executable: "C:/Program Files/Keysight/PathWave Test Sync Executive 2023B/TseService/bin/TseService.exe"
default_parameters:
 - "--kdi"
modes:
 - single
 - always
shared_secret: service_shared_secret
#
default_environment:
# Enable TSE Service Log Output
 - TSE_SERVICE_LOG_PATH=c:\\LogsTseService
# Enable HVI LOGGER - Required for troubleshooting (must be submitted with support requests)
# - HVI_LOGGER_LEVEL=Trace
# - HVI_LOGGER_FORCE_FLUSH=1
# - HVI_LOGGER_OUTPUT_PATH=c:\\Logs
```

NOTE Not that the TSE Service Log does not provide full information to enable troubleshooting, when reporting issues please enable the HVI LOGGER and include the output in the support request.

8.4.4. Shutting Down TSE Service

To shut down TSE Service, use the following command:

```
TseService --stop
```

This instructs the TSE Service to release any resources it has locked and then shut down.

The console output will look like this:

```
[2023-05-09 17:45:31.192] [hviOutput] [info] [mainTseService.cpp:24] [mainTseService] Starting TSE service, version: 2023
[2023-05-09 17:45:31.220] [hviOutput] [info] [TseServiceApp.cpp:375] [run] Stopping service pid 24836
[2023-05-09 17:45:31.231] [hviOutput] [info] [TseServiceApp.cpp:377] [run] Stop service pid 24836 result true
```

If there is an error in shutdown, try again.

NOTE It is possible to use Task manager to close TSE Service, but if you use this, TSE Service will not release any locks.

Task Manager should only be used as a last resort if TSE Service has crashed and it does not respond to the correct shutdown command.

Since it is not possible to run multiple instances of the TSE Service, to restart TSE Service you must first stop and then start it again following the manual or automatic execution described above.

8.4.5. Additional command line options

The following additional command line options are also available:

Option	Description
`--help`	Prints the available options.
`--listen_address <ip address>:<port>`	This is the address and port the TSE Server uses. You can alternatively provide this in the TSE Service configuration .yml file.
`--system_definition <path>`	Specifies the location of the **`system_definition.yml`** file
`--tse_config <path>`	Specifies the location of the **`tse_config.yml`** file
`--release_pxi_triggers <triggers>`	Release Pxi Triggers that are locked by an non-responsive client. Options are "`all`" or a comma separated list of triggers, such as "`1,2,3`". See *Handling Application Crash and Resource Locking*.

8.4.6. Checking the state of the TSE Service with a Client Application

You can check the state, mode and logs of a TSE Service instance with a client application.

NOTE For the log retrieval to be possible, make sure the environment Variable TSE_SERVICE_LOG_PATH is set on the host running the TSE service.

You connect to the TSE Service node and instantiate a TSE Service Client object. You can then use the properties in the object to query it. This is quick and convenient because it does not require a SystemDefinition.

The TSE Service client object enables you to access the state, mode and logs of the TSE Service node. You typically use a client application with systems in Free-Running mode.

The TSE Service client object exposes the following properties:

Property	Returns	Description
tse_service_mode	Returns the TSE Service mode.	The mode returned is the TSE Service node is running in. This is one of: `FREE_RUNNING`, `LEADER`, or `FOLLOWER`.
tse_service_state	This returns the TSE Service node state.	As the system starts up, it goes through a series of states. The state returned is the state the TSE Service node is in when it is queried. When the state returned is `RUNNING` you can execute your code. For a full list of the states, s ee the Python help files.
tse_service_log	Returns a string with the TSE Service Log.	This is the same TSE Service output that is sent to the console or file.

When the the Mode is correct and the State is RUNNING, the system is ready to run experiments.

The following snippet shows an example of how to check the state and mode of a Free Running host.

```
tse_service_host = 'tse://TestNode'
kdi_conn_options = "KdiUser=XXX,KdiPassword=****,KdiUrl=wss://localhost:9090/ws"
def wait_free_running_mode():
    tseServiceClient = keysight_tse.TseServiceClient(tse_service_host, kdi_conn_options)
    tse_service_state =  tseServiceClient.tse_service_state
    # Wait until we reach running state
    while tse_service_state != keysight_tse.TseServiceState.RUNNING:
        time.sleep(10) # sleep 10 seconds before retrying
        tse_service_state =  tseServiceClient.tse_service_state
        print(f"Current log on server: {tseServiceClient.tse_service_log}")
    # We are running, check mode
    tse_service_mode =  tseServiceClient.tse_service_mode
    if tse_service_mode != keysight_tse.TseServiceMode.FREE_RUNNING:
       raise Exception(f"Host {tse_service_host} is a {tse_service_mode}!")
# Wait here until the TSE Service on the 'tse_service_host' is running in Free-Running mode
wait_free_running_mode()
```

```
# At this point we know that the TSE Service in 'tse_service_host' is running in a FreeRunning mode.
# All the instruments defined in the tse_config.yml are opened and can be used
```

Example checking the status of a Leader Host

```
tse_service_host = 'tse://TestNode'
kdi_conn_options = "KdiUser=XXX,KdiPassword=****,KdiUrl=wss://localhost:9090/ws"
def wait_leader_mode():
    tseServiceClient = keysight_tse.TseServiceClient(tse_service_host, kdi_conn_options)
    tse_service_state =  tseServiceClient.tse_service_state
    # Wait until we reach running state
    while tse_service_state != keysight_tse.TseServiceState.RUNNING:
        time.sleep(10) # sleep 10 seconds before retrying
        tse_service_state =  tseServiceClient.tse_service_state
        print(f"Current log on server: {tseServiceClient.tse_service_log}")
    # We are running, check mode
    tse_service_mode =  tseServiceClient.tse_service_mode
    if tse_service_mode != keysight_tse.TseServiceMode.LEADER:
       raise Exception(f"Host {tse_service_host} is a {tse_service_mode}!")
# Wait here until the TSE Service on the 'tse_service_host' is running in Leader mode
wait_leader_mode()
# At this point we know that the TSE Service in 'tse_service_host' is running in a Leader mode.
# All the instruments defined in the tse_config.yml of all hosts are opened, system as it is defined in
# system_definition.yml of the host is initialized and ready to be used
```

Example checking the status of a Follower Host

```
tse_service_host = 'tse://TestNode'
kdi_conn_options = "KdiUser=XXX,KdiPassword=****,KdiUrl=wss://localhost:9090/ws"
def wait_follower_mode():
    tseServiceClient = keysight_tse.TseServiceClient(tse_service_host, kdi_conn_options)
    tse_service_state =  tseServiceClient.tse_service_state
    # Wait until we reach running state
    while tse_service_state != keysight_tse.TseServiceState.RUNNING:
        time.sleep(10) # sleep 10 seconds before retrying
        tse_service_state =  tseServiceClient.tse_service_state
        print(f"Current log on server: {tseServiceClient.tse_service_log}")
    tse_service_mode =  tseServiceClient.tse_service_mode
    # if LEADER, we know the node is not in the correct mode
    if tse_service_mode != keysight_tse.TseServiceMode.LEADER:
        raise Exception(f"Host {tse_service_host} is a {tse_service_mode}!")
    # For FOLLOWER, we need to keep waiting until it switches from FREE_RUNNING
    while tse_service_mode != keysight_tse.TseServiceMode.FOLLOWER:
       time.sleep(10) # sleep 10 seconds before retrying
       tse_service_mode =  tseServiceClient.tse_service_mode
# Wait here until the TSE Service on the 'tse_service_host' is running in Follower mode
wait_follower_mode()
# At this point we know that the TSE Service in 'tse_service_host' is running in a  Follower  mode.
# All the instruments defined in the tse_config.yml are opened and can be used
...
```

See *Accessing TSE Services* section below for more details and see *TSE Service Free-Running Mode* and *TSE Service Leader-Follower Mode* for more information on the different TSE Service modes and states.

8.4.7. Using Exceptions to check the status of TSE Service

You can also use exceptions to access the state, mode, and logs. For this you must create a SystemDefinition.

You can use exceptions if, for example, when you create a sequencer and there is a failure. You typically use exceptions in a system in Leader-Follower mode, in the remote SystemDefintion.

The `TseServiceException` exposes the same properties as a TSE Service Client object. Exceptions can be used in local or remote SystemDefintions.

The following snippet shows an example on how to use a Client Application with a remote SystemDefinition :

Opening instrument with KDI

```
kdi_user_option = 'KdiUser=hviuser1234,KdiPassword=hviuser1234password'
#
# Create system definition client connected to the TSE Service Leader using TSE Resource ID
try:
       system_def_client = keysight_tse.SystemDefinition("SystemDefinitionClient", "tse://host1", kdi_user_option)
       # Client SystemDefinition success only when TSE Service is in LEADER mode and RUNNING state
       tse_service_state = keysight_tse.TseServiceState.RUNNING
       tse_service_mode = keysight_tse.TseServiceMode.LEADER
#
except keysight_tse.TseServiceException as exc:
    tse_service_mode = exc.tse_service_mode
    tse_service_state = exc.tse_service_state
    tse_service_log = exc.tse_service_log
```

8.4.8. TSE Service Startup States and Modes

Each TSE Service node goes through a number of different modes and states during startup:

Modes

All nodes start in FREE_RUNNING mode when initializing and then switch to LEADER or FOLLOWER if that is how the system is configured.

States

All nodes pass through a series of states starting with BOOTING_UP. If the system is configured for Free-Running mode, once the node gets to the RUNNING state initialization stops and the node is ready for use.

If the system is configured for Leader-Follower mode, the mode changes to LEADER or FOLLOWER and performs further initialization until the state reaches RUNNING.

To see how the states and modes progress at start up, see the state diagrams in *TSE Service Free-Running Mode* and *TSE Service Leader-Follower Mode*.

For a description of the states see the *PathWave Test Sync Executive User Manual*.

8.5. Resource IDs for Accessing Remote Resources

The standard way to open resources (instruments, chassis, Sync Modules, etc) is to use a *VISA Resource ID*. There are other special resource identifier formats that enable access to resources distributed across the network:

- TSE and TSE-TCP Resource IDs.
- KDI Resource ID.

8.5.1. TSE Resource IDs to access Chassis, SSMs and TSE Service instances

In a system or user application with TSE Service, you can access chassis, *System Synchronization Modules* (SSMs) and TSE Service instances distributed across multiple hosts using the TSE Resource IDs. TSE Service and TSE Resource IDs can also be used in a Single chassis system.

The following table shows the 2 types of TSE Resource IDs available:

TSE Resource ID Format	Use Case	Example
`tse://<host>/<VISA_RESOURCE_ID>`	Requires TSE Service launched by KDI. Requires *KdiUser* and KdiPassword when used in the user application (see KDI Resource ID usage).	`For VISA_RESOURCE_ID = PXI0 : : 1 : : BACKPLANE` • `tse://myhost/PXI0 : : 1 : : BACKPLANE`
`tse-tcp://<host>:<port>/<VISA_RESOURCE_ID>`	Does not require KDI.	`For VISA_RESOURCE_ID = PXI0::CHASSIS1::SLOT10::INSTR` `tse-tcp://myhost:7587/PXI0::CHASSIS1::SLOT10::INSTR`

Where `<host>` can be specified using one of these forms:

1. Host Name (or Device Name), this is the recommended for most users. For example: `MyLabPc1`.
2. Full Host Name, For example: `MyLabPc1.NetworkDomain`.
3. IP address, this is not recommended, For example: `10.127.1.89`.

With TSE-TCP Resource IDs, the application must know the hostname and the TCP port used by TSE service to connect to or access resources controlled by that TSE Service, for example, to add a remote chassis or an SSM.

With TSE Resource ID, the service discovery is resolved by KDI and there is no need to specify upfront the TSE Service TCP port. As a result of using KDI, when using TSE Resource IDs in the client application you are required to specify the KDI User and Password options.

NOTE KdiUser, KdiPassword options are mandatory when using TSE Resource ID.

8.5.1.1. Accessing Chassis and System Sync Modules

To open/access the chassis, you must use the TSE or TSE-TCP Resource IDs explained above, depending on whether TSE Service is launched with KDI or not. For instance:

Chassis Resource ID specified in tse_config.yml in testNode2	TSE and TSE-TCP Resource ID
PXI0::1::BACKPLANE	tse://testNode2/PXI0::1::BACKPLANE tse-tcp://testNode2:8674/PXI0::1::BACKPLANE

For the System Sync Module the resource ID is derived from the corresponding chassis resource ID, for instance:

Chassis Resource ID specified in tse_config.yml in testNode2	Corresponding SSM TSE and TSE-TCP Resource ID
PXI0::1::BACKPLANE	tse://testNode2/PXI0::CHASSIS1::SLOT10::INSTR tse-tcp://testNode2:8674/PXI0::CHASSIS1::SLOT10::INSTR

Where *SLOT10* corresponds to the timing slot in the PXI0::1::BACKPLANE chassis (a 18-slot chassis).

The following snippet illustrates the above examples in code:

Opening Chassis & SSMs with TSE

```
# TSE Resource IDs rely on KDI infrastructure to resolve hosts and TCP ports
# Must specify a valid user registered in KDIS
kdi_user_option = 'KdiUser=hviuser1234,KdiPassword=hviuser1234password'
#
# Add Chassis using TSE Resource IDs
mySystemDefinition.chassis.add('tse://host1/PXI0::1::BACKPLANE',  kdi_user_option) #Chassis 1 in Host 1
#
# Add SSMs for each chassis with TSE Resource IDs (SSMs are opened already by TSE Service)
# the slot must be the timing slot of the PXI Chassis (for 18-slot chassis, it is slot 10)
primarySSM = mySystemDefinition.interconnects.add_sync_module('tse://host1/PXI0::CHASSIS1::SLOT10::INSTR',
 kdi_user_option)
```

Opening Chassis & SSMs with TSE-TCP

```
# Or add Chassis using TSE-TCP Resource IDs => must know the port the TSE Service is using
mySystemDefinition.chassis.add('tse-tcp://host1:8674/PXI0::1::BACKPLANE') # Chassis 1 in Host 1
```

```
#
# Add SSMs for each chassis with TSE-TCP Resource IDs (SSMs are opened already by TSE Service)
# must know the TCP port the TSE Service is using
# the slot must be the timing slot of the PXI Chassis (for 18-slot chassis, it is slot 10)
primarySSM = mySystemDefinition.interconnects.add_sync_module('tse-
tcp://host1:8674/PXI0::CHASSIS1::SLOT10::INSTR') #SSM for Chassis 1 in Host 1
```

8.5.1.2. Accessing TSE Services

In Leader-Follower mode, the client application creates a client system definition that connects to a Leader TSE Service. To establish this connection, the TSE or TSE -TCP Resources IDs must be used. The snippet below demonstrates this:

Create a Client System Definition with TSE

```
# TSE Resource IDs rely on KDI infrastructure to resolve hosts and TCP ports
# Must specify a valid user registered in KDIS
kdi_user_option = 'KdiUser=hviuser1234,KdiPassword=hviuser1234password'
#
# Create system definition client connected to the TSE Service Leader using TSE Resrource ID
sys_def = pyhvi.SystemDefinition("MyAppSystemDef","tse://host1", kdi_user_option)
```

Create a Client System Definition with TSE-TCP

```
# Create system definition client connected to the TSE Service Leader using TSE-TCP Resource ID
sys_def = pyhvi.SystemDefinition("MyAppSystemDef","tse-tcp://host1:8674")
```

8.5.2. KDI Resource ID to open instruments

KDI enables you to open or access instruments across the network using KDI Resource IDs. With the appropriate options, KDI also enables you to share the same instrument across applications and processes. For this, the first call to open a given resource launches an independent process where the hardware or service session is opened. Following calls to open the same resource with KDI connect to the existing session.

TSE Service makes use of KDI to open and initialize instruments at boot-up and get them ready to be used by the user application later. It is important to note that TSE Service performs the first open of the hardware resource, so applications do not need to apply any of the specific options required in the first call to open a resource with KDI.

NOTE When configuring TSE Service to open instruments at boot-up, you do not need to specify any of the options listed in the *First KDI open of a Resource*.

To open an instrument with KDI, the process is the same as without KDI, except:

1. Use the KDI Resource ID instead of the VISA one. The KDI Resource ID is built adding a prefix to the VISA Resource ID:
   - `kdi://<host>/<VISA_RESOURCE_ID>`.

2. For Authentication the initialization options must include:
   - *KdiUser* and *KdiPassword,* these are the user and password you set when you added a user (client) in KDIS, see explanation above on how to configure KDIS.
   - *KdiUrl* in general not needed, but it depends on the system configuration, see details below.
3. To open multiple KDI instances to the same Hardware you must include:
   - *AllowMultipleClientAttach=1*

If you intent to use the HVI Engine of the instrument with TSE API, the *First opening* of a KDI Resource, you must also include*:*

- *HviServer* => for example, for automatic IP and Port: `HviServer=HVITCP:[::]:0`

This first-time option is ignored in following open calls with KDI, as long as the specific resource is kept open.

For example, if the instrument is not open by TSE Service:

Opening instrument with KDI

```
myRemoteInstrument = keysight_ktmodule.KtModule('kdi://testNode1/PXI0::CHASSIS1::SLOT7::INSTR', 1,
1,'Simulate=0,DriverSetup=,KdiUser=user1234, KdiPassword=pass1234, AllowMultipleClientAttach=1,
HviServer=HVITCP:[::]:0, other_instrument_specific_options...')
```

You must always include the KDI user and password as defined in KDIS configuration when you open instruments in your application.

When the instrument is already opened by TSE Service, the user application must specify the KDI username and password and `AllowMultipleClientAttach=1`, other options are in general not needed, since they are passed by TSE Service:

Opening instrument with KDI

```
myRemoteInstrument= keysight_ktmodule.KtModule('kdi://testNode1/PXI0::CHASSIS1::SLOT7::INSTR', 1,
1,'Simulate=0,DriverSetup=,KdiUser=user1234, KdiPassword=pass1234, AllowMultipleClientAttach=1')
```

NOTE KdiUser, KdiPassword and AllowMultipleClientAttach options are mandatory.

NOTE All instances of the same instrument must use the same AllowMultipleClientAttach value, otherwise, you will get this error:

> *Could not start or attach to station service KtCornerstone 0.3.6914. Error message: Cannot connect to existing instance (PXI0::CHASSIS1::SLOT2::INSTR) because AllowMultipleClientAttach has not been set by both requestor (false) and instance (true).*

NOTE Simulate=false is the default option if not specified. For IVI compliant drivers, the option **"DriverSetup="** is mandatory after the IVI standard options and before instrument specific ones. "DriverSetup=" is not required for Python drivers, it is ignored.

8.5.2.1. KdiUrl initialization option

KdiUrl is an optional initialization option specific for instrument drivers, it must be specified as an initialization option for instruments when the KDI client configuration file does not include the *upstreammanager* entry, see *Configure KDI Clients to find KDIS* for more deatils:

Opening instrument with KDI

```
myRemoteInstrument = keysight_ktmodule.KtModule('kdi://testNode1/PXI0::CHASSIS1::SLOT7::INSTR', 1,
1,'Simulate=0,DriverSetup=,KdiUser=user1234, KdiPassword=pass1234, KdiUrl=wss://localhost:9090/ws,
AllowMultipleClientAttach=1, other_instrument_specific_options...')
```

NOTE If KdiUrl is required but not specified, you will get an error like:

Couldn't get test station manager URL through options parameter:Simulate=0,DriverSetup=,LogLevel=Info,KdiUser=hviuser1234,KdiPassword=hviuser1234,HviServer=HVITCP:[::]:0

8.5.3. Building the correct Remote Resource ID for multiple access

When specifying a Remote resource ID for multiple access, TSE, TSE-TCP or KDI, it is mandatory that all instances uses the same Resource ID format, in particular all Remote Resource IDs used in different instances that refer to the same Instrument must use the same VISA Resource ID when building the Remote Resource ID.

The table below show the different VISA Resource IDs supported for PXI Chassis and Instruments, and the one that is used by TSE Service that must be used by any application working with TSE Service:

Type	VISA Resource ID format	TSE Support
PXI Chassis Example: Chassis #1	PXI0::1::BACKPLANE	Supported
	PXI0::35-0::0::INSTR	Not Supported
	PXI35::0::0::INSTR	Not Supported
PXI Instrument (also System Sync Modules) Example: Instrument in Chassis #1 and Slot 12	PXI0::CHASSIS1::SLOT12::INSTR	Supported
	PXI0::CHASSIS1::SLOT12::INDEX0::INSTR	Not Supported
	PXI0::30-0.0::INSTR	Not Supported
	PXI30::0::0::INSTR	Not Supported

NOTE When the instrument or chassis is opened by TSE Service at boot up, the Remote Resource ID (KDI, TSE or TSE-TCP) used in the client application must include the same VISA_RESOURCE_ID as listed in the `tse_config.yml` file (or TSE Service Log). See *TSE Service Free-Running Mode* section for more details on instrument management. For instance, if the client application KDI Resource ID does not match the one used by TSE Service, you will get this error:

> *Failed to instantiate PXI0::CHASSIS1::SLOT2:: INDEX0:: INSTR. Error: <StdException>#std::runtime_error#Visa error 0xbfff000f. VI_ERROR_RSRC_LOCKED: Operation failed due to locked resource*

8.5.4. Using the HVI engines of remote instruments in user applications

In order to use TSE capabilities in the user application you must add the HVI Engines of the desired instruments in the application systemDefinition instance using the *HVI Engine unique ID* provided by the instrument drivers. HVI-capable instruments expose in the instrument drivers the *hvi* interface which includes properties with all the available HVI resources, in particular the *HVI Engine Unique IDs*, the following snippet illustrates how an HVI engine is added to the systemDefinition instance:

```
my_instrument = keysight_ktmodule.KtModule('PXI0::CHASSIS1::SLOT7::INSTR', False, False, InstrOptions)
#
my_sys_def = keysight_tse.SystemDefinition('Hvi') # optionally add Leader TSE Service Resource ID
#
my_sys_def.engines.add(my_instrument.hvi.engines.main_engine, "MyEngineAlias")
```

TSE has built-in multi-process and multi-host capabilities which enable a user application to add HVI Engines and exploit TSE capabilities of instruments opened in other processes and hosts. The information required for this is encoded in the *HVI engine Unique IDs*. An application in a different process or host only needs to know the *HVI Engine Unique ID* of a specific instrument to use it, regardless of the process or host where the instrument is opened.

By default, when an instrument is opened, the HVI Engine only supports in-host multi-process access implemented using shared memory for performance. To enable access from other hosts the *HviServer* option must be specified when opening the instrument, for example, for automatic IP and Port: "*HviServer=HVITCP:[::]:0*". The *HviServer* option opens a TCP server that allows TSE applications out of host to access and control the HVI Engine and HVI capabilities in that instrument. The following snippet illustrates how to open an instrument locally with multi-host access for the TSE capabilities:

Opening instrument with TCP server

```
my_instrument = keysight_ktmodule.KtModule('PXI0::CHASSIS1::SLOT7::INSTR', 1,
1,'Simulate=0,DriverSetup=,HviServer=HVITCP:[::]:0, other_instrument_specific_options...')
```

NOTE *"HviServer="* initialization option must be specified when opening instruments to enable multi-host access/control of TSE capabilities from a different host. This is required for instruments located in hosts different from the host where the application implementing the systemDefinition instance is executed.

8.5.5. Communication models in single process, multi process & multi host

HVI has three models of communication, single process, multi process, and multi host.

These models are leveraged depending on the environment of the application you are running, allowing you to write applications seamlessly through applications running in more than one process and in distributed hosts with devices.

- Shared memory is used to communicate data between the instruments and the client in a system with a multi process application.

- TCP is used to communicate data between the instruments and the client in a system with a multi host application.

The model of communication is chosen based on its expected performance:

- In process: Fastest, used when the application and the devices share the same process, makes direct function calls
- Shared memory: A bit slower, used when you open the instrument and add the engine in different processes, leveraging a shared memory pool to communicate operations.
- TCP: Slowest, used when you open the instrument in one host but add the engine in a different one, sends operations through the network.

The advantages of choosing shared memory over TCP are

- It is up to 8x faster than TCP loopback communications.
- It does not require additional port permissions.

Using shared memory

The mechanics of using shared memory are handled internally by Test Sync Executive. It is used in single host, multi process environments.

Shared memory will be prioritized whenever it is available.

Shared memory is enabled by default, allowing for faster communication when the client application and the server device are in the same host.

While no option is necessary, the processes running servers need to identify themselves.

By default, HVI uses the `process_id`, akin to passing `HviServer=HVISHM:[::]:<process_id>` (that is, the process id provided by the operating system) as the number that identifies a server in a process.

This will be reflected in the engine Ids and other device identifiers, for example.

You can override this with:

```
HviServer=HVISHM:[::]:<UserDefined_id>
```

To enforce a static engine Id. This override is useful when you want a constant ID across executions, whereas the `process_ID` changes between different runs.

The `UserDefined_id` must be unique for each process. In the case of KDI instruments it must also be unique for each instrument

For example:

```
my_instrument = keysight_ktmodule.KtModule('PXI0::CHASSIS1::SLOT7::INSTR', 1,
1,'Simulate=0,DriverSetup=,HviServer=HVISHM:[::]:1234, other_instrument_specific_options...')
```

Using TCP communication

You can also use TCP for communication between hosts, but this needs to be enabled explicitly.

The format options are:

- HVITCP:<host name>:<port>
- HVITCP:<host address>:<port>
- HVITCP:<ipv4 address>:<port>
- HVITCP:<ipv6 address>:<port>

To use an automatic port, you can use port 0:

- HVITCP:<name/address>:0

To use an automatic IP address with port 2000, you can use:

- HVITCP:[::]:2000
- HVITCP:0.0.0.0:2000

For example, for localhost:

- HVITCP:127.0.0.1:2000
- HVITCP:[::]:2000

For example, for Multi-Host:

- HVITCP:server1.mycompany.com:2000
- HVITCP:196.100.40.43:2000

For example:

HviServer=HVITCP:server1:2000

Using TCP and shared memory

You can use TCP and shared memory in a single instrument.

To use both TCP and shared memory you must list the TCP address, and optionally the shared memory address:

For example:

HviServer=HVITCP:<host Name>:<port>

HviServer=HVITCP:<host Name>:<port>;HVISHM:[::]:<UserDefined_id>

8.5.6. Handling Application Crash and Resource Locking

TSE Service manages the infrastructure (Chassis, clocking and System Synchronization Modules) and provides remote access to it. When a client application executes the `loadToHw()` command, it is TSE Service that reserves/locks the resources for the client application. If the client application crashes after the `loadToHw()` and before the `releaseHw()`, TSE Service will keep the resources locked and when trying to re-run the same or other application that uses the same resources, an error will occur. An example of resources that could stay locked are the PXI triggers or HVI Engines. To resolve this situation, the user needs to try one of the following approaches:

1. Execute from the command line:
   - **`TseService --release_pxi_triggers <triggers>`**
     - *where <triggers> can be:*
       - **`all`**
         - *This option will release all resources (triggers, engines) locked*
       - *or, a list of triggers to release (for example: **`0,1,2,3`**).*
         - *This option will release only the resources (triggers, engines) that were using the provided triggers*

   - Depending on the TSE Service running mode, the release will apply to:
     - In the case of Leader-Follower mode, resources (triggers, engines) locked by instrumentation controlled by the Leader TSE Service. This means the chassis, SSMs, and instruments included in the **`system_definition.yml`**
     - In the case of Free-Running mode, resources (triggers, engines) locked by instrumentation opened by the Free-Running TSE Service. This means the chassis, SSMs, and instruments included in the **`tse_config.yml`**
     - This operation is not allowed on TSE Service instances in follower hosts, you must execute it from the Leader.

2. Restart TSE Service (see the *TSE Service Execution* Section).
   - This option releases all the resources controlled by that TSE Service, but not resources locked by instruments opened in different processes by KDI.

For instruments opened directly by the client application with KDI, the resources are not released by these solutions. The user must terminate their respective processes manually, before running their client application.

NOTE Since the system where the TSE Services are running may be shared by multiple users, the user that needs to release resources needs to be very careful to not affect any other running applications with any of the solutions presented above.

8.6. TSE Service Free-Running Mode

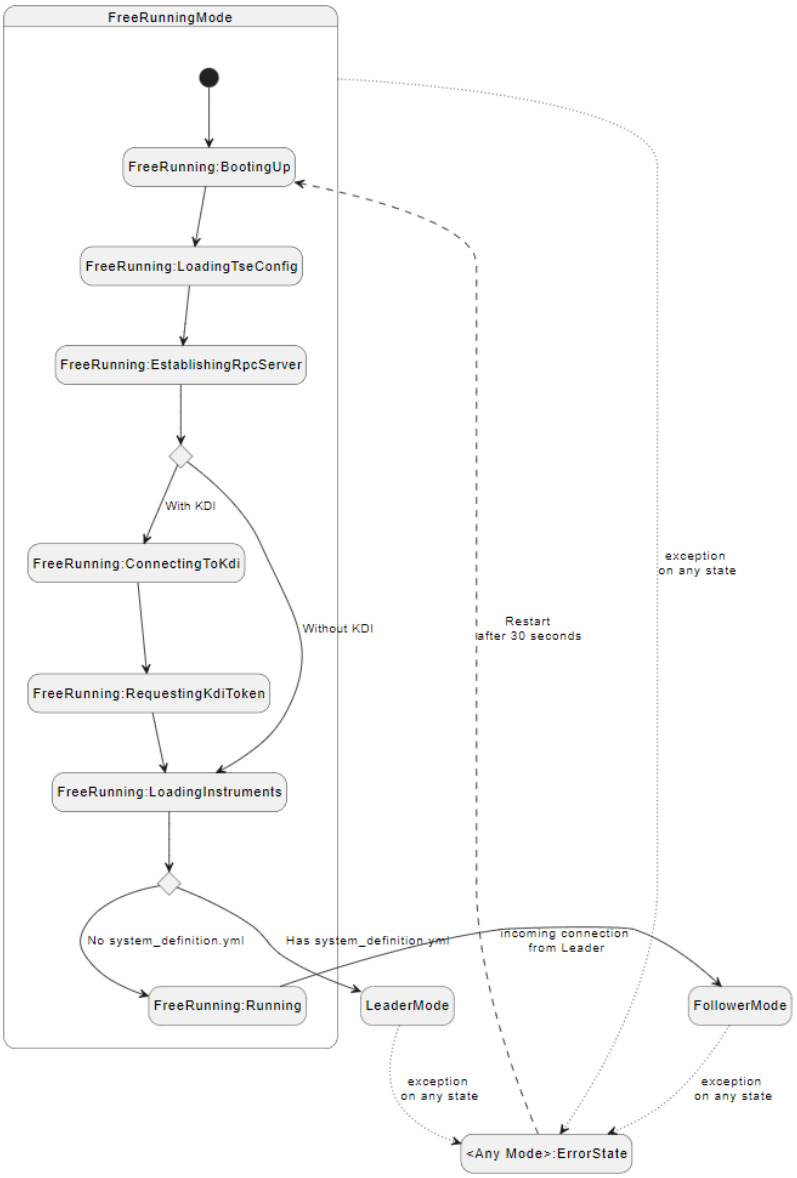
TSE Service in free-running mode is mainly intended for Multi-Host systems because it enables a client application to access resources distributed across multiple hosts. The client application is responsible for defining the topology of the system and initializing it, for instance, using the systemDefinition class to add chassis, SSMs, HVI engines, etc, and to run the system initialization.

User applications can:

- Access remote resources like chassis and *System Synchronization Modules* (SSMs) using TSE Resource IDs.
- Access remote instruments using KDI Resource IDs.
- Configure the system as required using the TSE and instruments API.

See the *Resource IDs* section above for details on how to access remote resources.

The following state machine diagram shows the TSE Service boot up sequence in Free-running mode:



For more details on the Leader-Follower mode, see the *TSE Service Leader-Follower Mode* section.

8.6.1. TSE Service configuration for Free-Running mode (tse_config.yml)

The Free-Running mode is the basic TSE Service configuration and is specified through the `tse_config.yml` file in each host. The TSE Service config file enables you to configure:

- The TCP port (this is strongly recommended if KDI is not used to launch TSE Service).
- Hardware resources: Chassis, *System Synchronization Modules* (SSMs) and Instruments.
- Also enables you to specify simulated hardware instances.

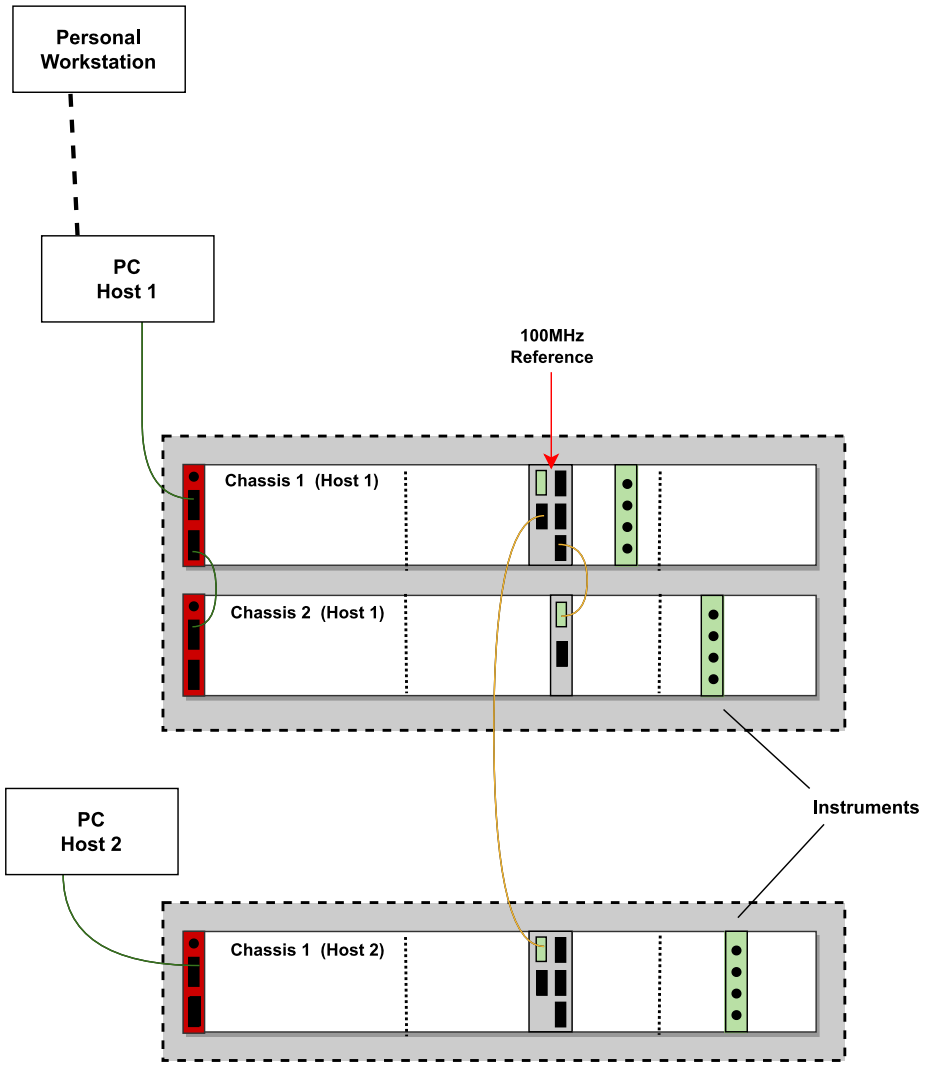
TSE Service opens the specified resources at boot-up, saving significant time when running the application.

The TSE Service config file is called `tse_config.yml` and it must be located in `C:\ProgramData\Keysight\PathWave Test Sync Executive 2023B\TseService\config`

For a full description of the `tse_config.yml` file options, see the *PathWave Test Sync Executive User Manual*.

8.6.2. Free-Running mode configuration example

The following diagram illustrates the 3 chassis Multi-Host system that is used in this example:



8.6.2.1. TSE Service Config File (tse_config.yml) with HW resources

All hosts in the system must have a TSE Service config file. For the example above, the default **tse_config.yml** that is included with the TSE installation works:

```
# Configure the port for the TSE Service server
# When launching TSE Service manually without KDI it is recommended to specify a port, for instance =>
0.0.0.0:8674
listen_address: 0.0.0.0:0      # Automatic IP and Port
#
# chassis list to be opened in the system boot
# SSM and HPRCS in each chassis are automatically detected and loaded in each chassis
chassis:
  - autodetect_all_pxi  # add all the PXI chassis
#
instruments:
  - autodetect_all_pxi    # open all detected PXI instruments at TSE Service boot-up
```

It is important to select a port that is available and permitted by your organization's IT policies. The auto detection of the port is recommended only when using KDI, since the KDI infrastructure enables it to resolve the actual port at runtime. When KDI is not used, the port tse-tcp URL must be used and the port specified to access any resource through TSE Service.

In the example above, we use the '`autodetect_all_pxi`' keyword to indicate TSE Service to open all detected PXI Chassis and instruments at boot-up time. This will speed-up opening hardware later in the user application. The SSMs and High Performance Clock Modules do not need to be specified explicitly because they are opened automatically with the chassis.

Initialization options for chassis or instruments can also be specified. For a full description of the **`tse_config.yml`** file options, see the *PathWave Test Sync Executive User Manual*.

NOTE For most instruments, except the M3xxxA family, KDI must be installed to allow the instrument to be opened by TSE Service and shared by the user application.

If you do not want to open all Chassis or instruments at boot-up, then replace the '`autodetect_all_pxi`' with the list of chassis or instruments to open. See the example below for simulation and remove the options "`Simulate=true, Model=xxxx`"

8.6.2.2. TSE Service Config File (tse_config.yml) for simulation

To work in simulation, you must specify all chassis and instruments that must be opened in simulation using the '`simulate=true`' and any other option required for the chassis or instrument to work properly in simulation. Wildcards are not supported for opening Chassis or Instruments in simulation. Resources must be specified using VISA_RESOURCE_IDs, do NOT use TSE, TSE-TCP or KDI Resource IDs.

tse_config.yml for Host 1

```
# configure the port for the server or keep it in auto "0.0.0.0:0" # Configure the port for the TSE Service
server
# When launching TSE Service manually without KDI it is recommended to specify a port, for instance =>
0.0.0.0:8674
listen_address: 0.0.0.0:0      # Automatic IP and Port
#
# chassis list to be opened in the system boot
# SSM and HPRCS in each chassis are automatically detected and loaded in each chassis
chassis:
  - PXI0::1::BACKPLANE: "Simulate=True, DriverSetup=Model=GenericPxieChassis,EnhancedTrigger=True"  #
Chassis 1 (host 1 - name: testNode1)
    ssm_options: "Simulate=true, DriverSetup=Model=M9033A"
  - PXI0::2::BACKPLANE: "Simulate=True, DriverSetup=Model=GenericPxieChassis,EnhancedTrigger=True"  #
Chassis 2 (host 1 - name: testNode1)
    ssm_options: "Simulate=true, DriverSetup=Model=M9032A"
#
instruments:
  - PXI0::CHASSIS1::SLOT10::INSTR: "Simulate=true, DriverSetup=Model=M5302A"
  - PXI0::CHASSIS2::SLOT15::INSTR: "Simulate=true, DriverSetup=Model=M5300A"
```

tse_config.yml for Host 2

```
# Configure the port for the TSE Service server
# When launching TSE Service manually without KDI it is recommended to specify a port, for instance =>
0.0.0.0:8674
listen_address: 0.0.0.0:0      # Automatic IP and Port
#
# chassis list to be opened in the system boot
# SSM and HPRCS in each chassis are automatically detected and loaded in each chassis
chassis:
  - PXI0::1::BACKPLANE: "Simulate=True, DriverSetup=Model=GenericPxieChassis,EnhancedTrigger=True"  #
Chassis 1 (host 2 - name: testNode2)
    ssm_options: "Simulate=true, DriverSetup=Model=M9033A"
#
instruments:
  - PXI0::CHASSIS2::SLOT16::INSTR: "Simulate=true, DriverSetup=Model=M5200A"
```

Note the *Simulate=true* and *Model* initialization options are specified for all chassis and instruments.

NOTE The same examples above can be used to specify the specific hardware instances that should be opened instead of all. For that just remove the *Simulate* and *Model* options.

NOTE It is recommended to not include hardware and simulated instruments together in the same tse_config file. Consequently, it is not recommended to specify simulated instruments and also specify the '`autodetect_all_pxi`' keyword, because that also opens any hardware resources detected.

8.6.2.3. KDI User when TSE Service is not launched by KDI

It is recommended to launch TSE Service automatically with KDI, but in those cases where that is not possible and TSE Service must be launched manually, the **`tse_config.yml`** file must include the KDI User & Password if the instrument section is included to open instruments (other than the M3xxxA family):

```
#.....
#
# Must specify KDI User and Password for the instrument section to open instruments for shared access
(default operation)
# Not needed for M3xxxA instrument family
kdi_user: user123456
kdi_password: pass123456
#
instruments:
#.....
```

When KDI User and Password are included in the `tse_config.yml` for a given TSE Service, all remote operations perform by this TSE service instance will be authenticated using the specified KDI User and Password.

8.6.3. User application with TSE Service in Free-Running mode

The user application must specify and initialize all the components they want to use in the application, by means of the SystemDefinition object and the TSE API:

After boot-up, and once the TSE Service is running and completed initialization (See TSE Service execution section), the user (or client) application must:

1. Open all instruments. You must use KDI Resource IDs as described above, except for M3xxxA product family.
2. Create a SystemDefinition and add the Chassis and SSMs using the TSE or TSE-TCP Resource IDs.
3. Complete the SystemDefinition configuration as usual:
   a. Specify the topology of the system by indicating the physical System Sync connections between the SSMs.
   b. Configured the clocking, Sync Resources, etc as needed.
   c. Add the Instrument Engines, etc.
   d. Trigger the system initialization => systemDefnition.initialize(...).
4. Create the Sequencer, HVI instance and rest of the application as usual.

The following snippet shows an example for a client application using TSE Service in free-running mode for the system and configuration examples above:

```
import keysight_tse as pytse
import keysight_ktmodule
#
# To access remote resources with the KDI infrastructure must use a valid user registered in KDIS
kdi_user_option = 'KdiUser=hviuser1234,KdiPassword=hviuser1234password'
#
#
**********************************************************************************************************
**********
# Check the TSE Service has completed the Free-Running initialization and is in the FreeRunning::Running
state
# Create TSE Service client object and query it
tse_service_host = 'tse://host1'
tseServiceClient = keysight_tse.TseServiceClient(tse_service_host, kdi_user_options)
tse_service_state =  tseServiceClient.tse_service_state
tse_service_mode =  tseServiceClient.tse_service_mode
while tse_service_state != keysight_tse.TseServiceState.RUNNING or tse_service_mode != keysight_
tse.TseServiceMode.FREE_RUNNING:
  # sleep 10 seconds before retrying
    time.sleep(10)
    # Check state
    tse_service_state =  tseServiceClient.tse_service_state
```

```
  # Check mode
    tse_service_mode =  tseServiceClient.tse_service_mode
# At this point we know that the TSE Service in 'tse_service_host' is running in a FreeRunning mode.
# All the instruments defined in the tse_config.yml are opened and can be used
#
#
****************************************************************************************************
***********
# Open instrument drivers using KDI since instruments are already opened by TSE Service (specified in the
tse_config.yml)
#
# Define init option including Kdi options
# HviServer and AllowMultipleClientAttach options are not needed because these instruments are already
opened by TSE Service
# KdiUrl is not needed if upstreammanager entry is included in the localhost kdi.yml configuration file.
InstrOptions =
'Simulate=0,DriverSetup=,LogLevel=Info,AllowMultipleClientAttach=1,KdiUrl=wss://localhost:9090/ws'
#
# Open instruments in host 1 using KDI Resource ID
module = keysight_ktmodule.KtModule('kdi://host1/PXI0::CHASSIS1::SLOT7::INSTR', False, False, f"
{InstrOptions},{kdi_user_option}")
modules.append(module)
module = keysight_ktmodule.KtModule('kdi://host1/PXI0::CHASSIS2::SLOT12::INSTR', False, False, f"
{InstrOptions},{kdi_user_option}")
modules.append(module)
#
# Define instrument in host 2 using KDI Resource ID
module = keysight_ktmodule.KtModule('kdi://host2/PXI0::CHASSIS1::SLOT12::INDEX0::INSTR', 1, 1, f"
{InstrOptions},{kdi_user_option}")
modules.append(module)
#
#************************************************
# Create Hvi instance
sys_def = pytse.SystemDefinition('Hvi')
#
# Add chassis using TSE Resource ID (chassis are opened already by TSE Service)
# to use TSE Resource IDs must include the KdiUser and KdiPassword
sys_def.chassis.add('tse://host1/PXI0::1::BACKPLANE', kdi_user_option) # host 1  chassis 1
sys_def.chassis.add('tse://host1/PXI0::2::BACKPLANE', kdi_user_option) # host 1  chassis 2
sys_def.chassis.add('tse://host2/PXI0::1::BACKPLANE', kdi_user_option) # host 2  chassis 1
#
# Add SSMs for each chassis with TSE Resource IDs (SSMs are opened already by TSE Service)
# to use TSE Resource IDs must include the KdiUser and KdiPassword
ssm_leader = sys_def.interconnects.add_sync_module('tse://host1/PXI0::CHASSIS1::SLOT10::INSTR', kdi_user_
option)
ssm_follower1 = sys_def.interconnects.add_sync_module('tse://host1/PXI0::CHASSIS1::SLOT10::INSTR', kdi_user_
option)
ssm_follower2 = sys_def.interconnects.add_sync_module('tse://host2/PXI0::CHASSIS1::SLOT10::INSTR', kdi_user_
option)
#
# ********************************************************************************************
# The rest of the application works as usual, no change for using TSE Service in free-running mode
# ********************************************************************************************
#
# ************************************
# Specify the topology
#
# Connect Leader Chassis to the 2 follower Chassis using SSMs SystemSync ports
ssm_leader.connectivity.systemsync_downstream[1].set_connection(ssm_follower1.connectivity.systemsync_
```

```
upstream[1])
ssm_leader.connectivity.systemsync_downstream[2].set_connection(ssm_follower2.connectivity.systemsync_
upstream[1])
#
# ************************************
# Configure clocking
#
# Configure Leader or primary SSM clock source
clockSource =   ssm_leader.clock_source
clockSource.set_mode(pytse.ClockingReferenceMode.EXTERNAL, 100e6)
#
# Assign Leader SSM as clock source for the whole system
sys_def.clocking.reference_source = clockSource
#
# ************************************
# Add HVI Engines
for index, module in enumerate(modules):
    main_id = module.hvi.engines.main_engine
    engine_alias = "Engine_"+ str(index) + "_" + str(main_id)
    sys_def.engines.add(main_id, engine_alias)
#
#
# Run system initialization
sys_def.initialize()
#
# ************************************************
# Create a sequencer
sequencer = pytse.Sequencer("sequencer", sys_def)
#
# define real-time sequence (Optional)
# ...
#
# compile the sequence
hvi = sequencer.compile()
#
# ************************************************
# Deploy and Execute on Hardware
#
# Deploy to hardware and initialize
hvi.load_to_hw()
#
# (optional) Run real-time sequence
hvi.run(hvi.no_timeout)
#
# When done - Release hardware
hvi.release_hw()
```

NOTE Your application using the TSE API can run in any host, you do not need to run the application in the hosts connected to the hardware.

NOTE If your application is used in a host with no hardware, then the TSE Service is not required to be running on this host. Only the Hosts with hardware connected to them require the TSE Service to be running.

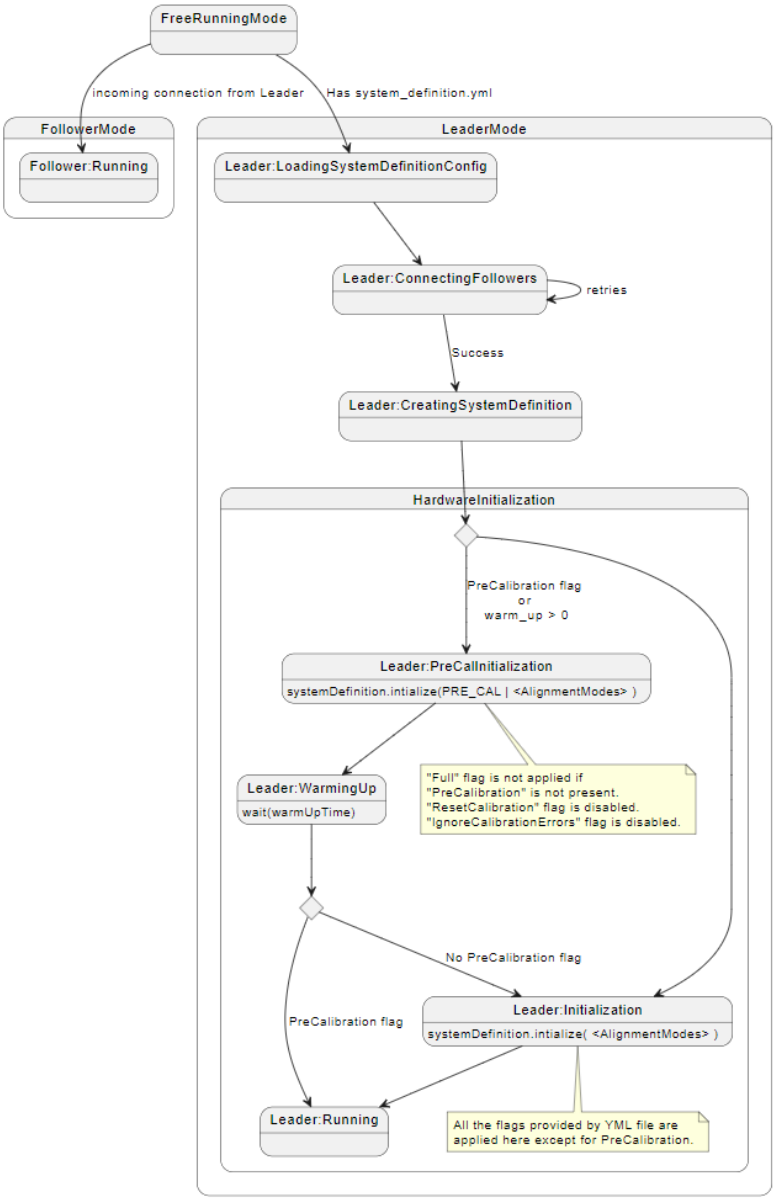
8.7. TSE Service Leader-Follower Mode

A system configured in Leader-Follower mode has a predefined system configuration that is fully initialized at start-up. This configuration is specified in the `system_definition.yml` configuration file located in one of the hosts, called the Leader TSE Service. The Leader TSE Service system definition can include resources distributed in other hosts running TSE Service, called Follower TSE Services. In the `system_definition.yml` configuration file the user can specify:

- System Topology.
  - This includes information for the chassis and interconnects (SystemSync connectivity) across them.
- Clocking configuration.
- Optionally instruments to include in the automatic start-up initialization.

In this mode, client applications specify the Leader TSE Service when creating the application systemDefinition instance and only need to specify the HVI Engines and related resources, but no need to specify topology and clocking because that's inherited from the Leader TSE Service. This is a simpler and quicker way to use a system because it enables client applications to exploit TSE real-time capabilities without having to first specify chassis, SSMs, clocks, etc. or having to wait for a full system initialization.

The state machine diagram below shows the initialization steps once TSE Service has completed the Free-Running initialization:



It is important to note that TSE Service always starts in **Free-Running mode** and performs the free-running initialization per the local **tse_config.yml** file. It is in this step where all TSE Service hosts open chassis, SSMs, Instruments, etc. depending on the local tse_config.yml file configuration. Then, only if the `system_definition.yml` file is found locally, this TSE Service switches to the **Leader mode.** In this mode it creates a SystemDefinition instance including all elements specified in the `system_definition.yml`, topology, clocking, and instruments if specified. Before creating the SystemDefinition, the Leader TSE Service connects to other TSE Services, the Follower TSE Services, that own the elements specified in the topology section. When a TSE Service received a connection request from the Leader, it switches from the Free-Running into the **Follower mode**, TSE Services accept only one Leader connection when in free-running mode.

8.7.1. TSE Service configuration for Leader-Follower

Leader-Follower systems are configured with 2 different configuration files:

- The TSE Service `tse_config.yml` configuration files, all hosts must have this file properly configured.
- The `system_definition.yml` configuration file, this file is only required in the Leader host.

This files can be found, or must be placed in this folder:

```
C:\ProgramData\Keysight\PathWave Test Sync Executive 2023B\TseService\config
```

8.7.1.1. tse_config.yml config file

The `tse_config.yml` file includes the TSE Service configuration common to all operating modes. All hosts in systems running TSE Service must have a `tse_config.yml`. See *TSE Service Free-Running Mode* section for more details

8.7.1.2. The Service `system_definition.yml` file

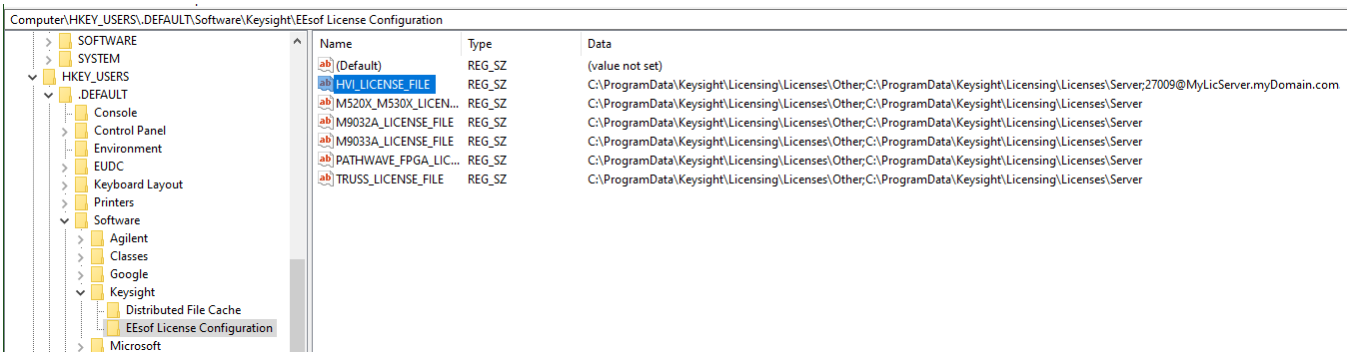
The `system_definition.yml` only exists in the Leader and defines the entire system configuration, including:

- System topology.
  - This include information of the Chassis and how they are connected by means of Sync Modules and SystemSync Connectivity.
- Clocking.
- Sync resources.
- [Optional] Instruments.

8.7.2. TSE Service Leader licensing requirements

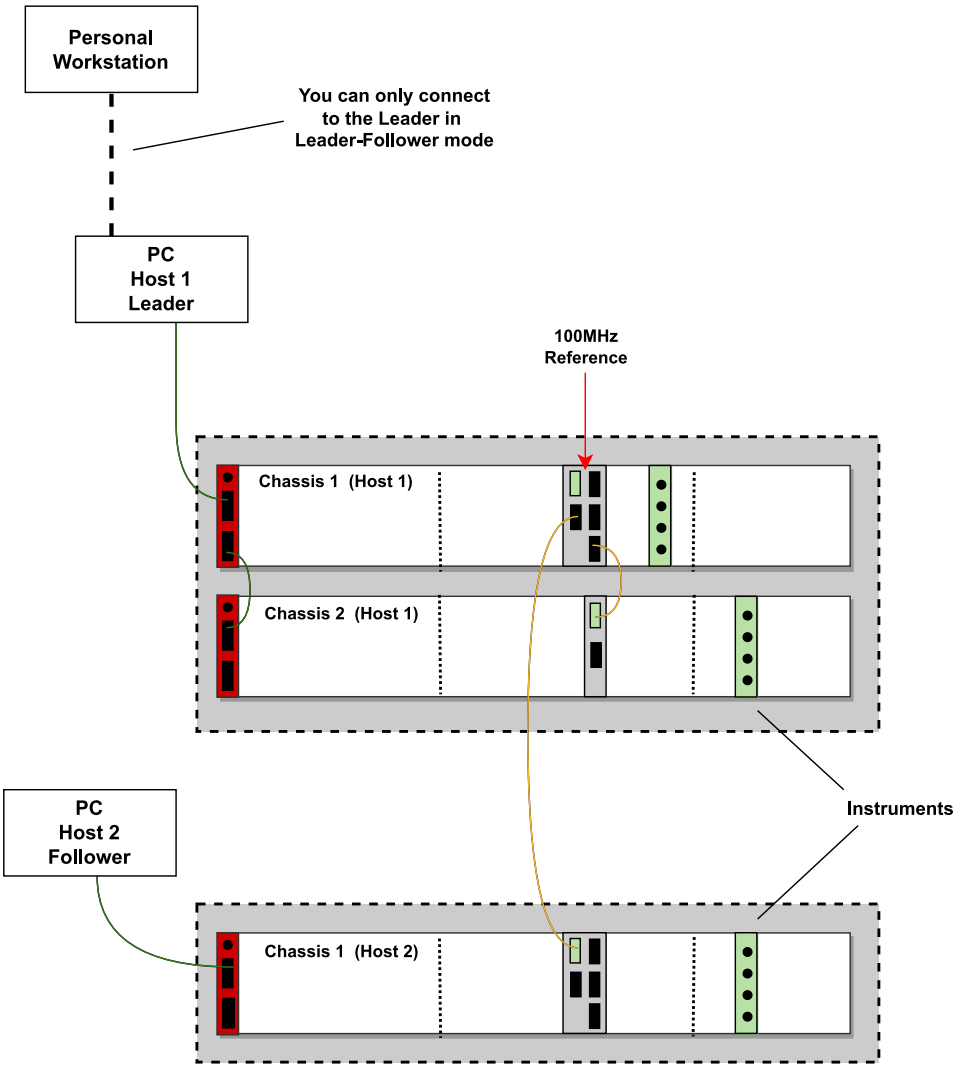
TSE Service to run in Leader Mode has the same licensing requirements as any client application. The number of TSE licenses required depends on the number of instruments, Sync Modules and Chassis. As a general guideline, one TSE license is required for each instrument and Sync Module included in the System Definition, or for a TSE Service Leader, specified in the `system_definition.yml` . See Licensing Documentation in the User Manual for more details on how to calculate the number of TSE licenses required for a given setup.

TSE Licenses must be properly configured using PathWave License Manager (PLM). Special care must be taken when using KDI to automate TSE Service launch at Windows boot-up, it is required that PLM is properly configured for the Windows' SYSTEM user, since KDI runs at Windows boot-up as this user. This is particularly relevant when using floating licenses configured in a different host. You must make sure the "*HKEY_USERS\.DEFAULT\Software\Keysight\EEsof License Configuration\HVI_LICENSE_FILE*" key lists all paths and servers to search for the TSE Licenses, the screenshot below is an example:



8.7.3. Leader-Follower mode configuration example

The Leader-Follower example will build on the example and configuration described in the Free-running mode Section. The example setup we will use is the same as used in the previous section:



8.7.3.1. tse_config.yml config file examples

Refer to the Free-Running example for details on how to configure the `tse_config.yml` .

8.7.3.2. The Service `system_definition.yml` file example

This an example **`system_definition .yml`** file to configure *host1* as TSE Service Leader for the setup described above:

```
### This in the Node the follower nodes to connect to
#
topology:
  - chassis: tse://host1/PXI0::1::BACKPLANE
    downstream:
      - 1: tse://host1/PXI0::2::BACKPLANE
      - 2: tse://host2/PXI0::1::BACKPLANE
#
clocking:
  # If this section is not included, TSE will assume SSM-Internal
  reference: ssm      # Allowed values are: ssm|chassis|hpc. Default is: ssm.
                      # We can assume the chassis and ssm to be the leader defined in the connections
earlier
  mode: external         # Allowed values are:  internal|external. Default is: internal.
  frequency: 100e6    # Allowed only with mode == external
#
sync_resources:
  - PXI_TRIGGER_0
  - PXI_TRIGGER_3
#
initialization_warmup_time_seconds: 30    # Time to wait between first initialization without Calibration
and final init with Calibration.
#
# Specify instruments to be added to the Leader SystemDefinition and initialized at boot-up
instruments:
  - inherit_from_tse_config # Initialize all instruments found for the Topology specified above per the tse_
config.yml files of each host
```

Instead of using the `inherit_from_tse_config` to include in the system initialization all instruments per the **`tse_config.yml`** in each host, it is also possible to list specify specific instruments in the **`system_definition .yml`** file. Note that if instruments are not opened already in the corresponding host per the **`tse_config.yml`** file, then the system initialization will be delayed waiting for the instrument to open. This is how the 3 instruments in our present example could be defined explicitly in the **`system_definition .yml`**:

```
...
#
# Specify instruments to be added to the Leader SystemDefinition and initialized at boot-up
instruments:
  - kdi://host1/PXI0::CHASSIS1::SLOT10::INSTR:
  - kdi://host1/PXI0::CHASSIS2::SLOT15::INSTR:
  - kdi://host2/PXI0::CHASSIS2::SLOT16::INSTR:
```

Note that KDI Resource IDs must be used for instruments in hosts other than the Leader, for the Leader host it is optional.

8.7.4. User application with TSE Service in Leader-Follower mode

When using TSE Service in leader-follower mode the user application creates a *SystemDefinition* instance which connects to the Leader TSE Service, a Client System Definition. To create a client

SystemDefinition instance you must specify the TSE or TSE-TCP Resource ID of the leader TSE Service:

- `SystemDefinition(<name>, <leader_address>)`
- `SystemDefinition(<name>, <leader_address>, <options>)`

When using TSE Resource IDs, the options must include the KDI User and Password. The following snippet demonstrates how to create a client SystemDefinition:

```
# To access remote resources with the KDI infrastructure must use a valid user registered in KDIS
kdi_user_option = 'KdiUser=hviuser1234,KdiPassword=hviuser1234password'
#
# Create system definition client connected to the TSE Service Leader
sys_def = pyhvi.SystemDefinition("Hvi","tse://host1", kdi_user_option)

# Create system definition client connected to the TSE Service Leader
sys_def = pyhvi.SystemDefinition("Hvi","tse-tcp://host1:8674")
```

If the connection fails, or the TSE Service is not ready in the Running state in the Leader mode, the operation will throw an exception with information on the TSE Service state if at least the connection to the TSE Service would be established.

The following code shows an application that connects to the Leader in a Multi-Host system in Leader-Follower mode:

```
import keysight_tse as pytse
import keysight_ktmodule
#
# To access remote resources with the KDI infrastructure must use a valid user registered in KDIS
kdi_user_option = 'KdiUser=hviuser1234,KdiPassword=hviuser1234password'
#
#
*****************************************************************************************************
************
# Create tse service client connected to the TSE Service Leader to check the status
#
tse_service_state = pytse.TseServiceState.NOT_STARTED
tse_service_mode = pytse.TseServiceMode.FREE_RUNNING
tse_service_log = ''
#
tse_service_host = 'tse://host1'
tseServiceClient = keysight_tse.TseServiceClient(tse_service_host, kdi_user_options)
tse_service_state =  tseServiceClient.tse_service_state
tse_service_mode =  tseServiceClient.tse_service_mode
while tse_service_state != keysight_tse.TseServiceState.RUNNING or tse_service_mode != keysight_
tse.TseServiceMode.LEADER:
  # sleep 10 seconds before retrying
    time.sleep(10)
    # Check state
    tse_service_state =  tseServiceClient.tse_service_state
  # Check mode
    tse_service_mode =  tseServiceClient.tse_service_mode
    if tse_service_mode==pytse.TseServiceMode.FREE_RUNNING and tse_service_mode == pytse.TseServiceMode.NOT_
STARTED:
        raise Exception(f"Host {tse_service_host} not available!")
    if tse_service_mode == pytse.TseServiceMode.FREE_RUNNING and tse_service_
mode==pytse.TseServiceMode.RUNNING:
        raise Exception(f"Host {tse_service_host} is not configured as a Leader!")
    if tse_service_mode == pytse.TseServiceMode.FOLLOWER:
        raise Exception(f"Host {tse_service_host} is a Follower!")
#
#
*****************************************************************************************************
******************
# Open instruments
modules = []
InstrOptions = 'Simulate=0,DriverSetup=,LogLevel=Info,AllowMultipleClientAttach=1'
#
module = keysight_ktmodule.KtModule('kdi://host1/PXI0::CHASSIS1::SLOT7::INSTR', false, false, f"
{InstrOptions},{kdi_user_option}")
modules.append(module)
#
module = keysight_ktmodule.KtModule('kdi://host1/PXI0::CHASSIS2::SLOT12::INDEX0::INSTR', false, false, f"
{InstrOptions},{kdi_user_option}")
modules.append(module)
#
module = keysight_ktmodule.KtModule('kdi://host2/PXI0::CHASSIS1::SLOT12::INDEX0::INSTR', false, false, f"
{InstrOptions},{kdi_user_option}")
modules.append(module)
#
#
```

```
#############################################################################################################
# Configure Client System Definition
#
# The topology is already defined in the Leader TSE Service,
# There is no need to add/configure chassis/SSMs, clocking, systemSync connectivity, etc
#
# Add Engines
for index, module in enumerate(modules):
    main_id = module.hvi.engines.main_engine
    engine_alias = "Engine_"+ str(index) + "_" + str(main_id)
     sys_def_client.engines.add(main_id, engine_alias)
#
# Call System initialize
# The initialization will only initialize the instruments for the Engines added in the system definition
# Infrastruture is managed and initialized by the Leader TSE Service
# This step only checks proper initialization for infrastructure
sys_def_client.initialize()
#
# ###########################################################################################################
# The rest of the application works as usual, no change for using TSE Service in free-running mode
# ###########################################################################################################
#
# ################################################
# Create a sequencer
sequencer = pytse.Sequencer("sequencer", sys_def_client)
#
# define real-time sequence (Optional)
# ...
#
# compile the sequence
hvi = sequencer.compile()
#
# ################################################
# Deploy and Execute on Hardware
#
# Deploy to hardware and initialize
hvi.load_to_hw()
#
# (optional) Run real-time sequence
hvi.run(hvi.no_timeout)
#
# When done - Release hardware
hvi.release_hw()
```

8.8. TSE Service In-Process execution

The TseServiceServer class allows the user to run a TSE service in the same process as the user application. Using this instead of the TSE executable can improve the performance. This is because we can run all the HVI engines in the same process as the application. This avoids using the shared memory mechanism for each engine. This said, there are some limitations that the user has to be aware of, listed in the user manual.

8.8.1. Usage

The TseServiceServer class is intended to control the instance of a TSE Service. This accepts parameters like the Tse Service executable. The creation of a Tse Service with default parameters would look like:

```
# Create a TseServiceServer
tse_server = keysight_tse.TseServiceServer("")
```

Additionally to the parameters, you can also directly pass the contents of the configuration files **`tse_config.yml`** and **`system_definition.yml`**

```
# Create a TseServiceServer with the contents of some configuration files
tse_config = Path("path/to/some/tse_config.yml").read_text()
system_definition = Path("path/to/some/system_definition.yml").read_text()
tse_server = keysight_tse.TseServiceServer("",  tse_config,  system_definition)
```

The accepted arguments supported by TseServiceServer is a subset of all the arguments supported by the TseService executable. The valid arguments are:

Option	Description	Example
listen_address	Override the listen address from the tse_config.yml file, the format must be <IP>:<port>	--listen_address 127.0.0.1:1234
system_definition	Specify the system_definition.yml file path. In case the path has spaces, it is mandatory to surround it in double quotes.	--system_definition "C:\Path with spaces\system_definition.yml"
tse_config	Specify the tse_config.yml file path. In case the path has spaces, it is mandatory to surround it in double quotes.	--tse_config "C:\Path with spaces\tse_config.yml"

The class TseServiceServer allows you to interact with the tseService instance, such as querying the status of the service, get the port address being used, and manually manage when to stop the tseServiceServer. You can read more about it in the API documentation.

9. System Troubleshooting

This section contains explanations to troubleshoot common setup issues or resolve common error messages returned during the execution of your application.

If an error occurs while you are running HVI, an error message is typically displayed on the console. This message can originate in either:

- HVI itself.
- One of the components that HVI controls.

In the second case, HVI outputs a message identifying the instrument and process that generated the error, prefixing the error itself by the string `Product error` for easy identification.

NOTE If an error message includes `Product Error`, to ensure the best and fastest service, report the problem directly to the support representatives for the relevant product, for example Chassis, HPRCS, SSM or Instruments.

9.1. Troubleshooting tips

When you are using more than 1 chassis, you must:

- Use the latest chassis driver and firmware.
- Specify the connections between the chassis in the TSE API.

Chassis numbering

- Ensure your chassis are numbered from 1 upwards.

  The PXI standard does not permit chassis to be numbered as 0. If this happens, it indicates there has been an incorrect installation of the firmware, PXI chassis driver, software, or PXI resource manager.

Ensure you are using correct firmware and software components

- For PathWave Test Sync Executive to work correctly, the PXI chassis, firmware, driver, software, and PXI Resource Manager must be all be installed correctly, regardless of the chassis vendor.

  Compatibility requirements for PathWave test Sync Executive are listed at Instrument Software and Firmware Requirements for KS2201A.

Using non-Keysight chassis with PathWave Test Sync Executive

- Keysight recommends you use PathWave Test Sync Executive with Keysight chassis. It is It is possible to use non-Keysight chassis with the following limitations:

a. Only a single PXIe chassis is supported if you are using a non-Keysight chassis. Multi-chassis operation requires the recommended Keysight PXIe Chassis.
b. The proper PXIe resource manager and chassis VISA driver installation is required.
c. PathWave Test Sync Executive has not been validated with non-Keysight PXIe chassis, if you encounter any issues, contact Keysight support.

Using non-Keysight chassis with Keysight Instruments and PathWave Test Sync Executive

- Check the documentation of each PXI instrument that you are using with PathWave Test Sync Executive, to ensure they comply with the instrument limitations on compatibility with non-Keysight chassis or controllers.

9.2. Generic troubleshooting procedure

Common courses of action to resolve errors are described in the following tables. If the error is not listed or the proposed action does not solve the problem, follow these steps:

1. Rerun the system initialization forcing `FULL` initialization => `my_system.initialize(keysight_tse.AlignmentModes.FULL)`
2. If the error persists, power cycle the complete hardware setup and run the system initialization forcing `FULL` initialization.
3. If the error still persists, and as a very last resort, consider running a `FULL` initialization with `RESET_CALIBRATION` => `my_system.initialize(keysight_tse.AlignmentModes.FULL | keysight_tse.AlignmentModes.RESET_CALIBRATION)`

   WARNING: Resetting the system calibration in this way will in turn also force you to rerun the *System Warm-up and Calibration* procedure described above, which for some instruments requires to redo the User calibration. So observe extreme caution when doing this to avoid costly and time-consuming recalibration.

9.3. Error messages and troubleshooting guide

The following tables collect common error messages and explain the most common causes of the underlaying issue and how to solve it.

9.3.1. System Setup Errors

Error message	Explanation	Common causes and possible fixes
`Chassis with number n not found`	Adding chassis `n` to the `SystemDefinition` using `add_chassis` fails.	PXIe cable to chassis `n` is not plugged in. Chassis `n` is not powered up. Chassis `n` was not discovered during PC host boot-up.
`IVI ERROR: library:KtM9032x_64.dll, error_code:x - Invalid session ID (VI_NULL).` `No error message could be retrieved. Please check the documentation`	Adding an SSM to the `SystemDefinition` using `add_sync_module` fails.	There is no SSM in the SSM slot of at least one chassis. One of the SSM being used in not recognized as an HW by the PC/chassis controller. Check Connection Expert to confirm if this is the root cause. Try power cycling to solve it. The SFP of at least one SSM is open when it shouldn't be.
`ERROR: Only leader Sync Module can be configured as clock source; leader cellini is connected to chassis n`	Setting the clock source to be an SSM other than the SSM in leader chassis `n.`	Set the leader SSM (with no SSM connected to its upstream System Sync port) to be the clock source.
`ERROR: Chassis Clock Source can only be used in the chassis where the leader Sync Module is connected; clock source chassis is n and leader Sync Module chassis is n`	Setting the clock source to be a chassis other than the leader chassis `n.`	Only the leader chassis hosting the leader SSM can be set to be a clock source
`Chassis only supports 10MHz external reference. Please set an external reference frequency of 10MHz when using a chassis clock source as a reference`	The frequency of the external reference for the chassis is incorrectly set using `set_mode()` when using the chassis as a clock source.	Set the frequency of the external reference clock to 10 MHz.
`Chassis n doesn't support High Performance Clock Source`	Leader chassis `n` is not an M9046A with HPRCS. This is required to set the clock source to HPRCS.	Replace leader chassis `n` with a chassis containing an HPRCS.

Error message	Explanation	Common causes and possible fixes
`Kt9546x: wait reference clock set mode fails.: Please check the physical connection between Chronos and its clock source`	Incorrect external reference clock for the HPRCS in the leader chassis.	Reference clock set to the incorrect frequency. HPRCS Ref In input not connected. Double check the external reference clock source going into the HPRCS.
`Hardware Error: Kt9546 Pll is not locked`	M9546 fails to lock to the external reference clock when using HPRCS as clock source.	Power cycle the chassis to reset the M9546 HPRCS Make sure to install the latest M9546 HPRCS driver available on www.keysight.com. Check if the HPRCS OCXO is damaged by using the chassis SFP (see M9046 chassis documentation). If the issue persists, please open the M9546A SFP and try running a Utilities->Self Test. If this does not solve the issue, please share the self-test results with customer support. If the OCXO appears to be damaged or the issue persists, please reach out to Keysight support at https://support.keysight.com/ to send the chassis for repair.
`Chassis's route error from trigger line XX in bus segment YY to trigger line 0 in trigger bus segment 1: IVI-C driver error[xxx]: The chassis generated an error: INTERNAL_SW: Unexpected internal software error. Error code (hex): 0xbffa0018 from method set_route_ext_bus_mode`	The chassis driver you are using is out of date	Please install the latest chassis driver from the chassis webpage on www.keysight.com
`Error opening IVI driver for chassis M9044 - PXI::BACKPLANEIVI ERROR: library:KtPxiChassis_64.dll, error_code: xxxx - Invalid session ID (VI_NULL)... Please check the documentation`	The EEPROM of the M9046 chassis is programmed as M9044 by mistake	Please reach out to Keysight support at https://support.keysight.com/ to receive instructions about how to correctly reprogram the chassis EEPROM

9.3.2. Initialization errors

Error message	Explanation	Common causes and possible fixes
`operation "Link Initialization" failed: FdsConnectorAdapter::initializeConnectivity: Module info: Chassis n, Slot 10, HVI version x, Firmware version y. Product error: The SystemSyncUp_x8 port is not hooked up. FdsConnectorAdapter::initializeConnectivity: Module info: Chassis m, Slot 10, HVI version x, Firmware version y. Product error: The SystemSyncDown_x8 port(s) is not hooked up`	FDS System Sync link initialization failed.	System Sync cable connecting the SSMs in chassis `m` and `n` is missing (downstream `s` of SSM `m` to upstream of SSM `n`).
`operation "LinkAlignment" failed: Could not align link CHASSIS#n:SLOT#10::PxiBackplane#0:Port#s:Tx -> CHASSIS#n:SLOT#m::PxiBackplane#0:Port#0:Rx. FdsConnectorAdapter::finishAlignment: Module info: Chassis n, Slot m, HVI version x, Firmware version y. Product error: Required Pipe Select for DstarB of -5 exceeds expected max value of 7. Latency of bank 10. Latency count 15 Could not align link CHASSIS#n:SLOT#m::PxiBackplane#0:Port#0:Tx -> CHASSIS#n:SLOT#10::PxiBackplane#0:Port#s:Rx. FdsConnectorAdapter::finishAlignment: Module info: Chassis n, Slot 10, HVI version x, Firmware version y. Product error: FDS measured latency count larger than expected fixed latency count`	FDS DSTARB/C link initialization between SSM `n` and the instrument in chassis `n` slot `m` failed.	Power cycling the hardware setup should fix this. If the error persists then it might be a hardware error that you should report to Keysight.
`Hardware Error: Chassis n is not locked to SyncModule clock`	–	Power cycling the hardware setup should fix this. If the error persists then it might be a hardware error that you should report to Keysight.

Exception in RpcHandler:"Chassis.PXI <?>::<?>::BACKPLANE.reserve" version:"<?>": Chassis's multiple reservation error for trigger line <?> in trigger bus segment <?>: IVI ERROR: library:KtMTrig_64.dll, error_ code:-1074118122 - The chassis generated an error: PXI Trigger Manager - An attempt was made to reserve a line already reserved by this client. Error code (hex): 0xfffffffb from method pxi<?>_set_reservation_multiple	PXI triggers (line, segment) are a shared resource and can only be used by one entity at once	Trying to get hold of an already reserved PXI trigger. A possible solution is described in HandlingApplicationCrashandResourceLocking. Failing that, power cycling the HW also clears all reserved/locked resources

Initialization errors (continued)

The following SSM messages are of similar nature and are all preceded by the chassis number the SSM is in, the failing function call and the HVI and FW versions:

Error message	Explanation	Common causes and possible fixes
`Product error: PLL unlocked after MMCM input phase adjustment`	The leader SSM in chassis n fails to lock to the Ref1Out reference from the chassis front panel. The leader SSM in chassis n fails to lock to the external reference clock.	Cable is not connected, or it is connected to the incorrect SMA output. The external reference clock is missing.
`Product error: Unable to lock to REF_IN. Verify clock configuration and cabling and re-run the test. Refer to the KS2201A PathWave Test Sync Executive System Setup Guide for further guidance`	Rear 10 MHz chassis n REF IN BNC input is not present. Leader SSM n Ref In input is not connected.	The external reference clock has the incorrect frequency. This is usually because of missing external clocks, cabling issues, or an incorrect frequency setting. Check all of these.
`Product error: Unable to lock to System Sync FWD_CLK. (Same as above ...)`	-	Follower SSM `n` upstream System Sync connection fails.
`Product error: Unable to lock the internal OCXO source. (Same as above ...)`	-	This indicates a hardware issue because we should always be able to lock to the internal OCXO.
`Product error: Unsupported reference frequency. Only 10MHz and 100MHz supported`	-	Incorrect reference frequency specified.

Initialization errors (continued)

Error message	Explanation	Common causes and possible fixes
`Product error: LMKCLKout11 phase measurement out of range. Verify clock configuration and cabling and re-run the test. Refer to the KS2201A PathWave Test Sync Executive System Setup Guide for further guidance`	The SSM might have applied an incorrect calibration, or a calibration calculated for a different configuration.	Check if the HPRCS cable is correctly connected to the SSM Ref In Check if you are using an incorrect SMA output instead of the HPRCS output. Try to run a FULL initialization with RESET_CALIBRATION => my_system.initialize(keysight_tse.AlignmentModes.FULL | keysight_tse.AlignmentModes.RESET_CALIBRATION). If this does not solve the issue, try power cycling the hardware and rerunning the test. If the error persists then it might indicate a hardware issue.
`Product error: PLL unlocked after LMKCLKout11 phase adjustment. (Same as above ...)`		
`Product error: PLL unlocked after DSTARA/CK100_STM phase adjustment. (Same as above ...)`		
`Product error: FPGA_10M phase alignment to CLK100_STM failed. (Same as above ...)`		
`Product error: PXIe_SYNC100 alignment to Sync100_Base failed. (Same as above ...)`		

Initialization errors (continued)

Error message	Explanation
`The base frequency required by application (XXX Hz) is not compatible with system alignment base frequency (YYY Hz) as configured by TSE Service. In case you are using a sub-topology, make sure to set the flag 'support_client_subtopologies' to true. Please update TSE Service system_definition.yml to include the application configuration/requirements.`	In a Leader-Follower mode, the client application, that is connected to a TSE Service leader node, may have a Sync Base frequency that is not an integer multiple of the Sync Base frequency of the system represented by the TSE Service.

Common causes and possible fixes

Possible reasons that can lead to the calculation of an Sync frequency by the client application that is incompatible with that of the system are the following:

1. Engines added only to the client application with distinct core frequencies
2. The client application is using a subtopology of the system defined on the Leader system_definition.yml and this leads to a distinct physical propagation delay

To resolve this issue, the user needs to update the `system_definition.yml` on the Leader node. Depending on the reason above, the user will need to:
For 1:

- add the instruments used in the client application in the `system_definition.yml` file
- or, add the sync base frequency required by the application in the `non_hvi_system_clocks` list of the `system_definition.yml`

For 2:

- enable the support for subtopologies by specifying in the `system_definition.yml` file the following option:

```
support_client_subtopologies: true
```

9.3.3. Rare SSM errors (preceded by the chassis number the SSM is in, the failing function call and the HVI and FW versions)

Error message	Explanation	Common causes and possible fixes

`Product error: PXIe_CLK10 phase measurement out of range. Verify clock configuration and cabling and re-run the test. Refer to the KS2201A PathWave Test Sync Executive System Setup Guide for further guidance`
 	
Other similar messages:
 	
`Product error: FPGA_10M phase detection failed. (Same as above ...)`
`Product error: FPGA_10M phase unstable. (Same as above ...)`
`Product error: PXIe_CLK10 phase out of range. (Same as above ...)`
`Product error: PXIe_CLK10 phase adjustment failed. (Same as above ...)`
	-	These errors are usually resolved by rerunning the test. If the error persist, try to power cycle the hardware setup. If it still persists, please contact Keysight.

9.4. Frequently Asked Questions (FAQs)

Q: I cannot see any Analog output from the M9046 Chassis Front Panel (FP) 2.4 GHz output connector. How can I solve it?

A: Please make sure that your application is using the TSE API code lines explained in the section "Enabling the chassis Analog clock" to activate the 2.4 GHz reference on the M9046 FP. Moreover make sure you use the HV API code lines exaplined in the section "Configuring the Analog Clock Source in Instruments" to make sure your instruments are set to use the external 2.4 GHz reference instead of any reference internal to the instrument.

Q: I am receiving errors that point to a missing clock cable connection but the cable seems to be there. How can I troubleshoot it?

A: Please inspect the cable connection and the cable connectors. Not fully plugged cables or damaged connectors can cause the error to show up even if the cable seems to be there at a first look.

Q: How can I troubleshoot the error message "Hardware Error: Kt9546 Pll is not locked"?

A: 0) Power cycle the chassis to reset the M9546 HPRCS and check if it solves the problem 1) Please install the latest M9546 driver from the chassis tech support webpage on www.keysight.com. 2) By

using the PXIe Chassis SFP please set the clock source to HPRCS and check if the "Reference clock status" is Locked. 3) If the issue persists, please open the M9546A SFP and try running a Utilities->Self Test. If this does not solve the issue, please share the self-test results with customer support 4) If the HPRCS clock source cannot be locked from the chassis SFPor the self-test do not solve the issue as explained in steps 2)-3), please contact Keysight support at https://support.keysight.com/ to check if the HPRCS OCXO is damaged and may need replacement or any other possible solution for this issue.

10. How to Use HVI Logs to Report an Issue

PathWave Test Sync Executive comes with an integrated logger that you can use for troubleshooting.

The HVI Logger is aimed at producing information that is useful for support engineers. It provides information that is additional to the Sequence Representation output.

The logger has the following features:

- The level of logging is configurable.
- You can force flush messages.
- The output can be configured to go to the console or to an output file.
- You can configure the logger from environment Variables or in a `.env` configuration file.
- You can instruct some instruments to produce logs.

The logger can produce the following levels of logging information, where each level also includes all the information in the levels below it:

Logger level	Description
Trace	Produces trace information that is useful to support engineers.
Debug	Produces debug information that is useful to support engineers. This level also provides the Sequence Representation output.
Info	Produces generally useful information.
Warning	Logs anything that can potentially cause application oddities, but are automatically recovered.
Error	Logs any errors that are fatal to an operation, but not the service or application.
Fatal	Logs any errors that forces a shutdown of the application.
Off	Does not log anything.

10.1. Logger Configuration

The logger is configured with environment Variables. The following table describes the Variables:

Environment Variable	Values	Description
`HVI_LOGGER_LEVEL`	• Trace • Debug • Info • Warning • Error • Fatal • Off	This value indicates the level of information printed to the log. The information printed out contains the information for the level specified and all of the levels below it. For example, if the level is set to Debug, all messages except Trace are printed to the log. By default, the level is set to Error, so only Error and Fatal are printed.
`HVI_LOGGER_OUTPUT_PATH`	Any existing valid path in your system, For example: `C:\tmp`	This Variable disables console output and tells the logger to save the log to a file at the specified location. The file with the log messages is called: **`HVILog_hviLogger_[num1]_[num2].log`** , where **`num1`** is the date and time, and **`num2`** is the thread ID.
`HVI_LOGGER_FORCE_FLUSH`	`1` or `0`	This Variable forces the log messages to be flushed to the output every time a message is logged. Enable this if you want to troubleshoot a program that is crashing, so that all messages before the crash shall be written. Do not enable this option in any other cases, because it impacts the performance of the execution.
`HVI_LOGGER_EXTENDED`	"`*`", "`ALL`", or a comma separated list. For example: `M9032,M9546`	This Variable enables the logging output of instruments managed by HVI. An output file for each instrument is generated in the path specified with **`HVI_LOGGER_OUTPUT_PATH`** . The file is saved as: `{MODEL}_{Chassis Slot for M903x}_{date}.log` See the section *Logger Extended mode Supported Instruments* for a list of supported instruments.

NOTE By default the configuration for the logger is:

- Logging level: Error.
- Output: console.
- Force flush: disabled.
- Logger Extended: disabled.

10.2. .env Configuration File

The logger configuration can be also configured from a `.env` file. The configuration values are stored in the file as KEY=VALUE pairs and you can use `#` for comments.

The `.env` file must be located in the same folder as the HVI script to be executed. HVI parses the `.env` file and sets all the environment Variables found for that script.

The following shows an example `.env` file:

.env

```
# The hvi logger level: Trace, Debug, Info, Warning, Error, Fatal, Off.
HVI_LOGGER_LEVEL=Fatal
#
# Set this parameter to write the logs to a file instead of being printed to the console
HVI_LOGGER_OUTPUT_PATH=C:\tmp\hviLogs
#
# Set this parameter to force flush the log every new line instead of doing it at the end.
# This helps you to identify the line of code before a crash.
HVI_LOGGER_FORCE_FLUSH=0
#
# Activates the Logger for all HVI controlled instruments. The supported models are the
# System Synchronization Modules (M9032,M9033), the High Performance Reference Clock Source, (M9546)
# or "ALL"
HVI_LOGGER_EXTENDED=ALL
```

10.3. Logger Extended mode Supported Instruments

PathWave Test Sync Executive can control a number of different instruments. The environment Variable `HVI_LOGGER_EXTENDED` activates logging output from the instruments that support it. The way the logs are produced depends on the instruments, some instruments produce individual log files whereas other instruments combine log files together into a single file.

The supported models for release 2023B are:

Model	Description
M9546x	High Performance Reference Clock Source
M9032, M9033	System Synchronization Modules

NOTE Some Instruments, like the Keysight M5000 PXIe family for example, might include native logging facilities that cannot be controlled by PathWave Test Sync Executive, for more information, see your instrument documentation.

10.4. Recommended Logger settings for contacting support

If you require support for PathWave Test Sync Executive, a log file will help the support team to rapidly diagnose any problems.

If you want to contact support, first generate a log with the following settings:

- `HVI_LOGGER_LEVEL=Trace`
- `HVI_LOGGER_FORCE_FLUSH=1`
- `HVI_LOGGER_OUTPUT_PATH=C:\Logs` or another path [1]

1 The path must be an existing valid path.

Edition 2023C_U0_00, February, 2024
Printed in USA

KEYSIGHT TECHNOLOGIES

KS2201-90009
www.keysight.com