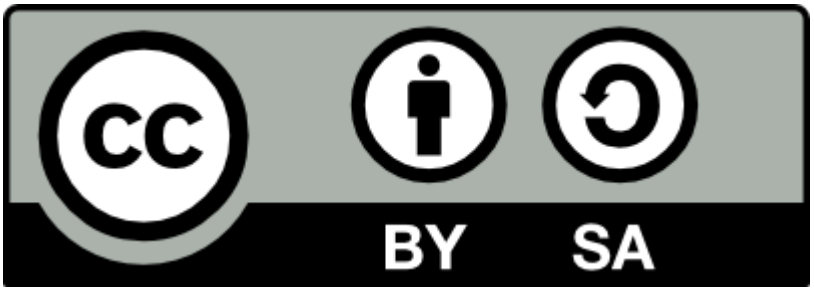
SiLA 2 Part (A) - Overview, Concepts and Core Specification

Release v1.1 - 19 March 2022

http://www.sila-standard.org/

Notices

License



SiLA 2 Version History

For the version history, please refer to Part (A) (Current Version).

Refer to the Structure of the SiLA 2 Specification for details about how the different documents are related.

SiLA 2 Working Group Members

Authors, co-authors and contributors to the SiLA 2 standard.

Name	Role	E-Mail
Daniel Juchli	Work Group Leader	daniel.juchli@sila-standard.org
Maximilian Schulz	Member	maxxsschulz@gmail.com
Armin Engesser	Member	armin.engesser@siobra.de
Florian Meinicke	Member	florian.meinicke@cetoni.de
Georg Hinkel	Member	georg.hinkel@tecan.com
Jonas Austerjost	Member	jonas.austerjost@sartorius.com
Kok Hien Gan	Member	KokHien.Gan@thermofisher.com
Jeff Bonevich	Member	jeff.bonevich@thermofisher.com
KV Rao	Member	kv.rao@thermofisher.com
Lukas Bromig	Member	lukas.bromig@tum.de
Mark Doerr	Member	mark.doerr@uni-greifswald.de
Markus Meser	Member	markus.meser@siobra.de
Michael Osthege	Member	m.osthege@fz-juelich.de
Mike Groezinger	Member	mike.groezinger@siobra.de
Mitko Georgiev	Member	mkgeorgiev@gmail.com
Patrick Oberthür	Member	patrick.zirker@tu-dresden.de
Ritter Daniel	Member	d.ritter@ritter-pt.ch
Robert Soeldner	Member	Robert.Soeldner@Sartorius-Stedim.com
Sebastian Hans	Member	sebastian.hans@tu-berlin.de
Sebastian Schärer	Member	sebastian.schaerer@roche.com
Stefan Koch	Member	koch@equicon.de
Dominic Lüthjohann	Contributor	dominic.luetjohann@labforward.io
Enrique Mireles	Contributor	enrique.mireles@labforward.io

Julian Luebke	Contributor	julian.luebke@labforward.io
Bernd Huf	Contributor	BHuf@hamilton.ch
Jérémy Zogg	Contributor	
Julian Lübke	Contributor	luebke@cubuslab.com
Oskari Vinko	Contributor	oskari@unitelabs.ch
Ricardo Gaviria	Contributor	ricardo@unitelabs.ch
Samuel Bernhard	Contributor	sbernhard@hamilton.ch
Uwe Kindler	Contributor	uwe.kindler@cetoni.de
Christian Waltenspiel	Technical Writer	christian.waltenspiel@web.de
Patrick Ley	Technical Writer	okhofsk@gmail.com
Alexander Reeder	Observer	alexander.reeder@wega-it.com
Andy Mitchell	Observer	Andrew.Mitchell@unilever.com
Benjamin Schulz	Observer	benjamin.schulz@ipa.fraunhofer.de
Christian Ulmer	Observer	c.ulmer@campus.tu-berlin.de
Christina Mavreas	Observer	christina.mavreas.ipa@gmail.com
Daniel Marquard	Observer	marquard@iftc.uni-hannover.de
Dewang Sharma	Observer	dewang.sharma@paa-automation.com
Ekkehard Görlach	Observer	ekkehard.goerlach@novartis.com
Mario Blattner	Observer	mario.blattner@wega-it.com
Mark Auty	Observer	Mark.Auty@unilever.com
Mark Porr	Observer	porr@iftc.uni-hannover.de
Pierre-Luc Satin	Observer	pierre-luc.satin@med.uni-goettingen.de
Simon Seidel	Observer	simon.seidel@campus.tu-berlin.de
Simon Stumm	Observer	simon.stumm@Sartorius.com
Stefan Maak	Observer	stefanmaak@freenet.de
Thomas Thaler	Observer	thomas.thaler@wega-it.com
Tim Meyer	Observer	tim.meyer@med.uni-goettingen.de

Niklas Mertsch	Observer	niklas.mertsch@stud.uni-goettingen.de
Bart van der Schoot	BoD Member	bart.vanderschoot@sila-standard.org
Burkhard Schaefer	BoD Member	b.schaefer@bssn-software.de
Erwin Althof	BoD Member	erwin.althof@novartis.com
Freundel Matthias	BoD Member	matthias.freundel@ipa.fraunhofer.de
Haike Suering	BoD Member	Haike.Suering@perkinelmer.com
Jason Meredith	BoD Member	jason.meredith@tecan.com
Oliver Peter	BoD Member	oliver.peter@idorsia.com
Patrick Courtney	BoD Member	patrick.courtney@sila-standard.org
Rob Harkness	BoD Member	rob.harkness@sila-standard.org
Thomas Frech	BoD Member	thomas.frech@sila-standard.org
Tom Kissling	BoD Member	tom.kissling@roche.com

SiLA 2 Working Group Organization

Please refer to Part (A) (Current Version).

SiLA 2 Roadmap

Please refer to Part (A) (Current Version).

SiLA 2 Adoption

Please refer to Part (A) (Current Version).

Status of The SiLA 2 Specification

Please refer to Part (A) (Current Version).

Table of Contents

Abstract

SiLA's mission is to establish international standards which create open connectivity in lab automation. SiLA's vision is to create interoperability, flexibility and resource optimization for laboratory instrument integration and software services based on standardized communication protocols and content specifications. SiLA promotes open standards to allow integration and exchange of intelligent systems in a cost-effective way.

SiLA 2 is based on open, well-established communication protocols and defines a thin domain-specific layer on top of these, consisting of common concepts and a vocabulary / taxonomy.

What's New in SiLA 2 Version 1.1

SiLA 2 Version 1.1 introduces a new ·Connection Method·, the ·Server-Initiated Connection Method·. This new feature enables a few exciting new use cases for SiLA:

- Connect lab instruments to the cloud
- Solve issues of accessing lab instruments, that reside in isolated lab networks, from a corporate network
- Simplify security certificate management for encrypted and secure connections

The New ·Server-Initiated Connection Method·

- Is a new SiLA 2 capability
- Allows the ·Connection· to be initiated by the instrument (the ·SiLA Server·), not only the controlling software (the ·SiLA Client·)
- Implemented secure encryption and authentication, so that instruments can be firewalled
- Enables standards-based IoT in the lab (no more vendor-specific approaches)
- Allows placing applications in the cloud or in the corporate data center
- All aspects of SiLA 2 (such as ·Features·, ·Commands·, …) will continue to work in a transparent manner
- Is supported by SiLA 2 reference implementations, no additional work for developers

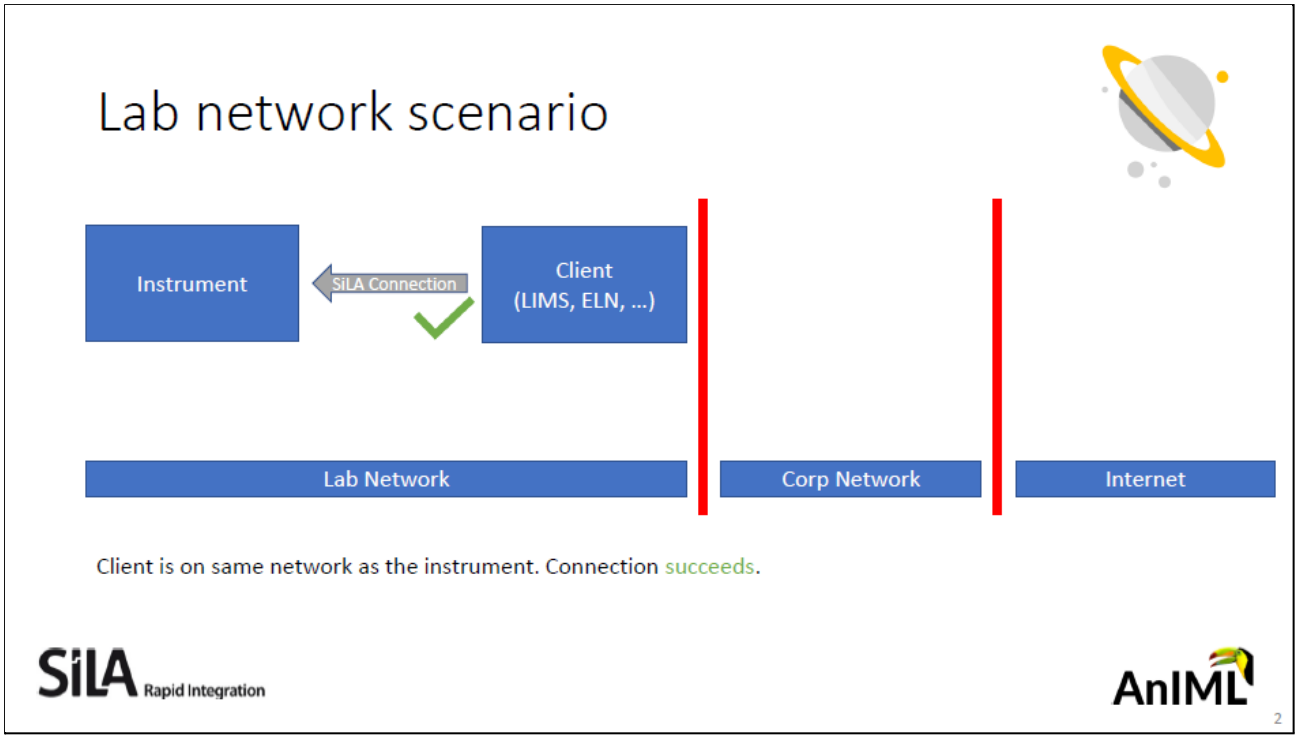
The Need for a Cloud-Native Approach

When communicating with an instrument, it must be reachable via the network. Connecting to the instrument may be difficult when using

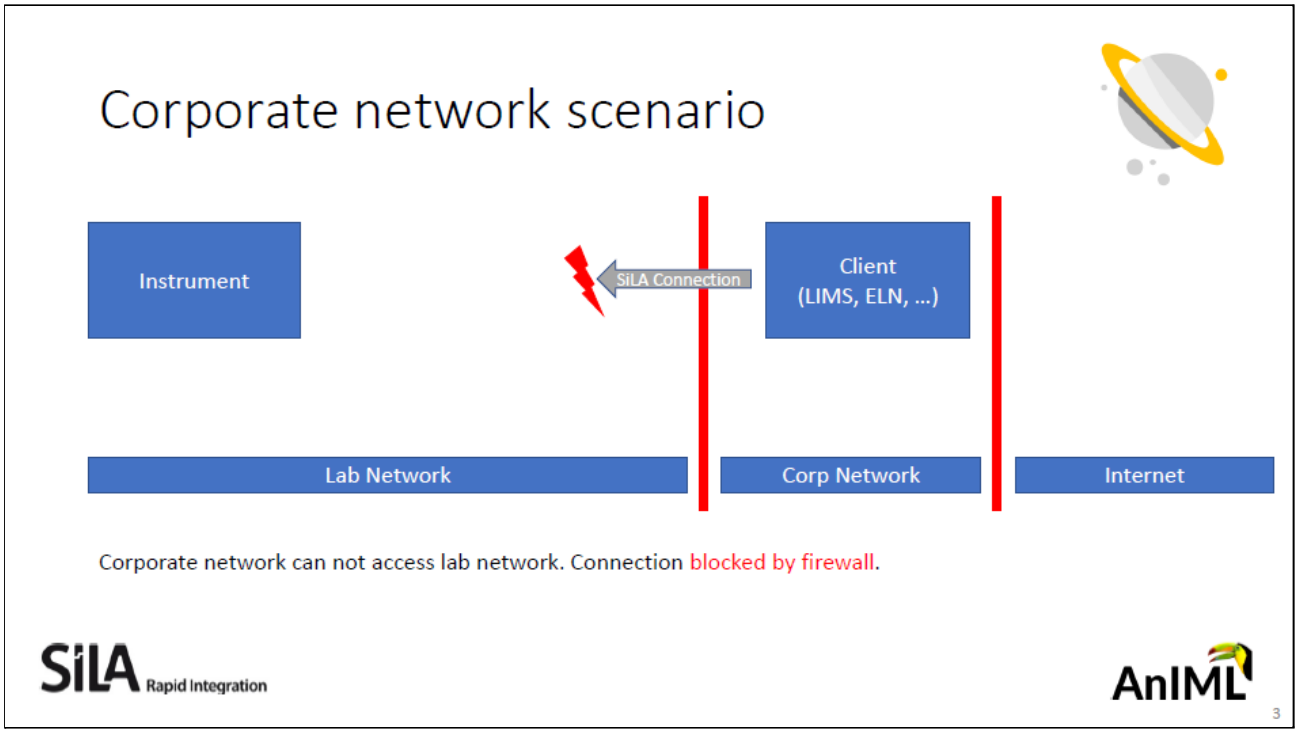
- Isolated lab networks
- Cloud-hosted applications

The Scenarios

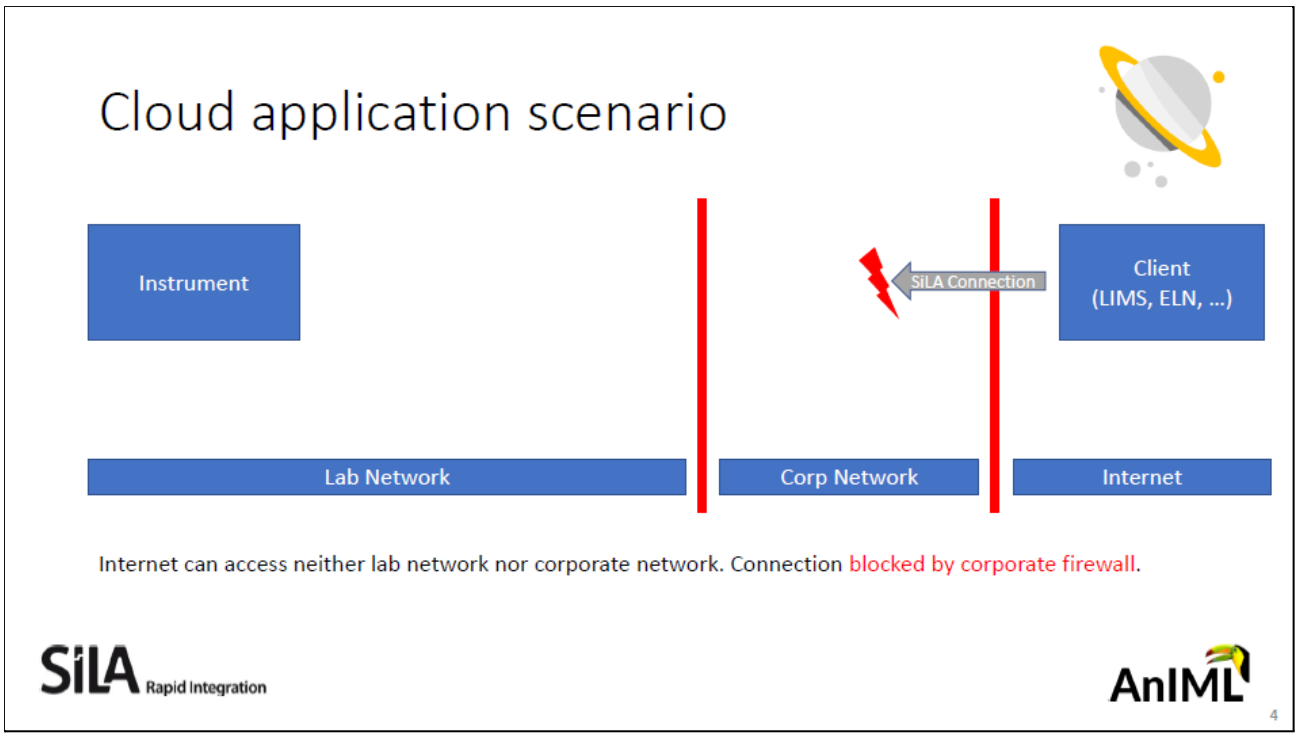
The Lab Network Scenario



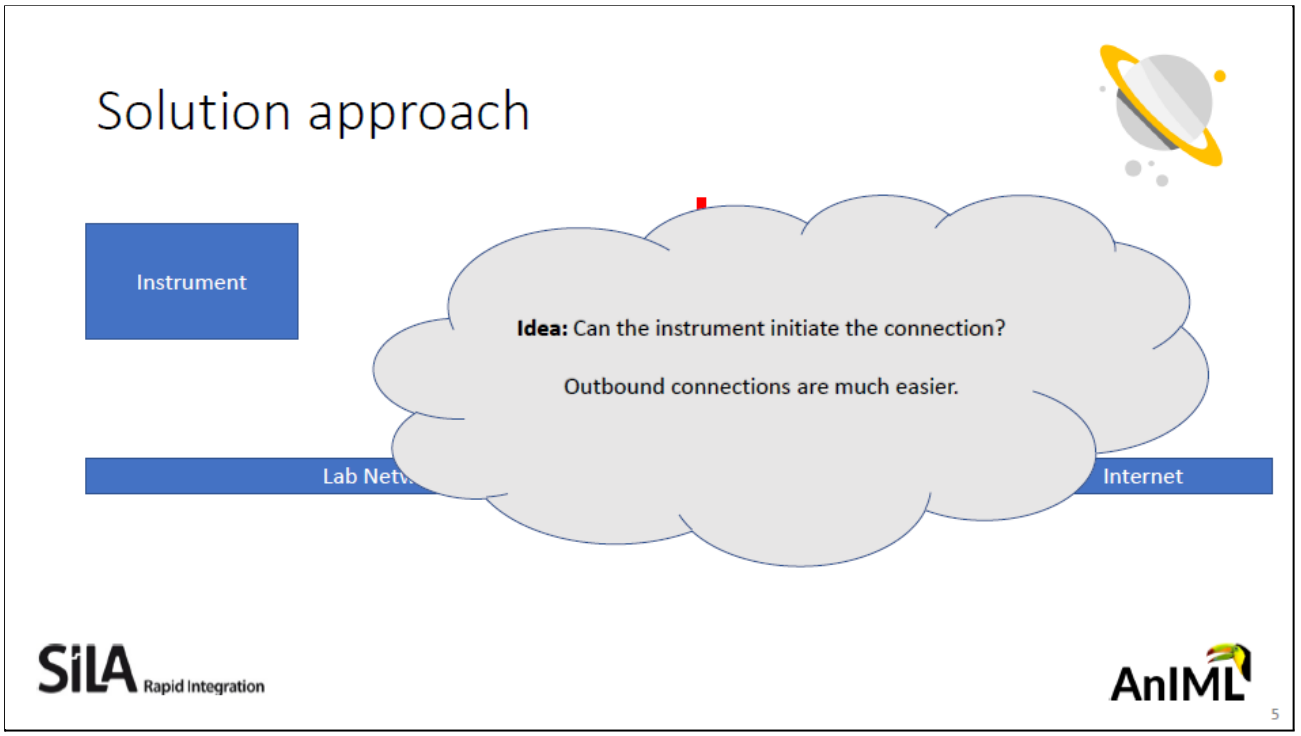
Corporate Network Scenario



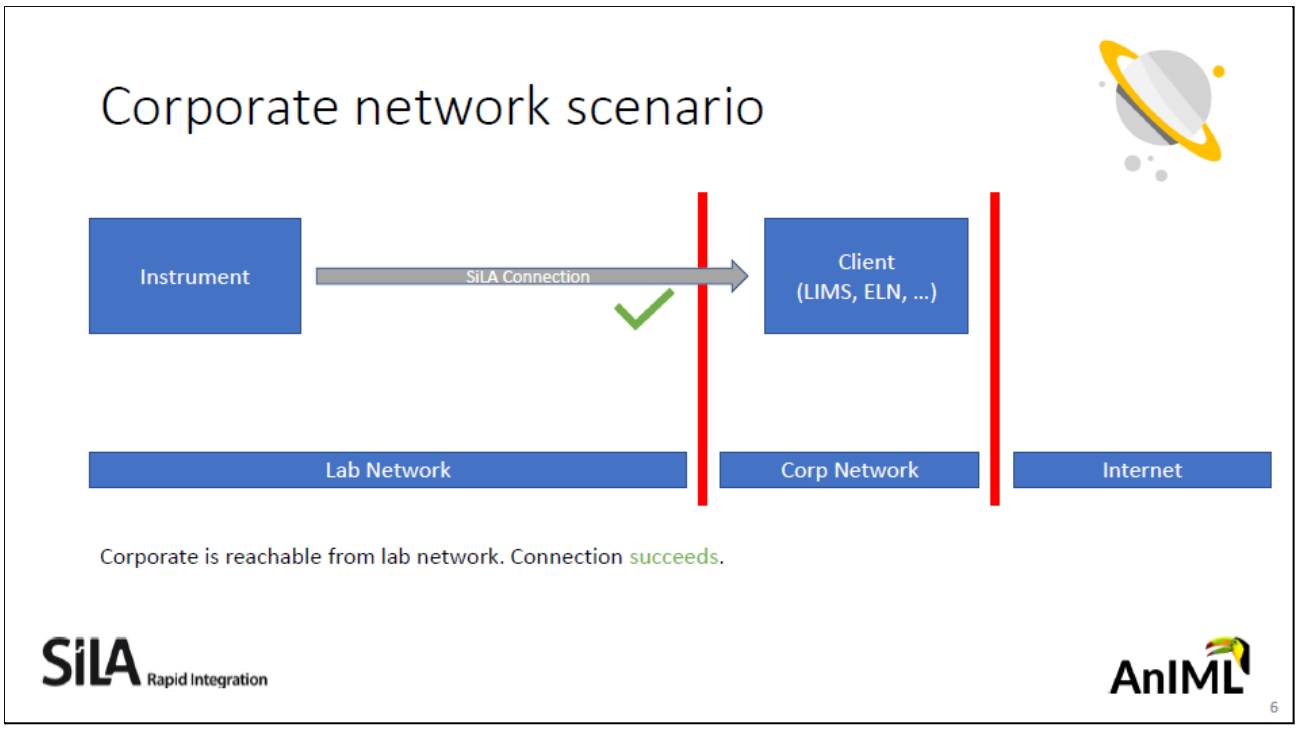
Cloud Application Scenario



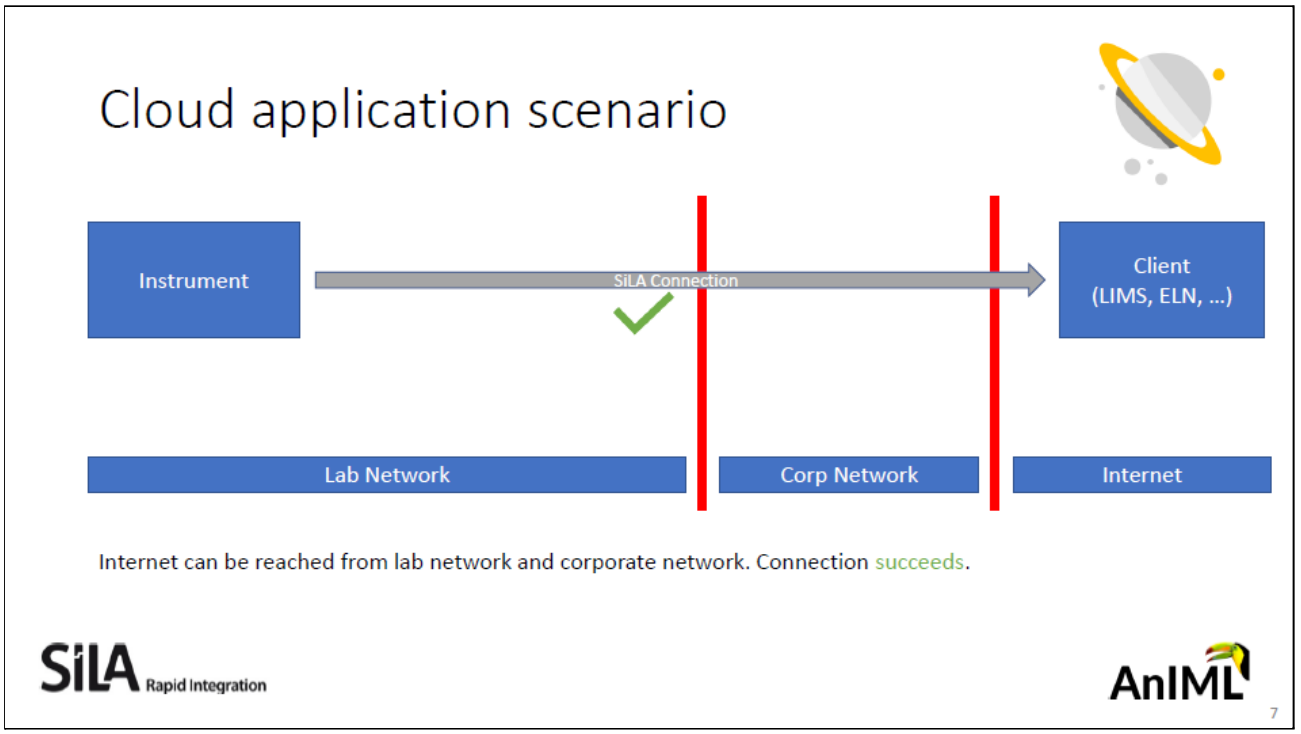
The Solution Approach



Corporate Network Scenario



Cloud Application Scenario



Technical Background of the New ·Server-Initiated Connection Method·

The SiLA cloud connection opens a virtual channel from the lab into the cloud.

The benefits of this approach are:

- No additional infrastructure needed for device vendors and lab environment
- Easy to develop – same technology stack. Cloud functionality can be added as lib for each language platform. Once added, can be used transparent for any feature
- Standard gRPC and HTTP/2 protocol connection handling and security models can be used

This approach makes it easy to connect to the cloud, meeting the same security requirements for the connection, such as authentication and encryption of the data. It also meets the protocol buffer messages specification and serialization to ensure interoperability between different platforms and vendors.

The Cloud SiLA client endpoint can be public HTTP/2 gRPC server endpoints, using standard TLS certificates for secure connections. For lab device vendors, this makes life much easier as there are no special requirements for the lab components.

The java reference implementation includes working examples for all aspects covered by new SiLA 2 ·Server-Initiated Connection Method·.

Like all SiLA 2 reference implementations, the reference of the new ·Server-Initiated Connection Method· can be found at https://sila2.gitlab.io/sila_base/ and https://gitlab.com/SiLA2.

Introduction

The SiLA Organization provides with the SiLA 2 Standard a framework for the exchange, integration, sharing, and retrieval of electronic laboratory information. These standards define how information is packaged and communicated from one system to another, setting the language, structure and ·SiLA Data Types· required for seamless integration between systems. SiLA standards support good laboratory practice and the management, delivery, and evaluation of laboratory services, and are recognized as the most commonly used in the world.

The SiLA 2 Specification specifies interoperability schemes that allow laboratory devices and services to communicate with each other. It also defines the process of further developing the standard and its assets.

SiLA was formed in 2008. Its standards have been implemented numerous times. In 2015, The SiLA Organization decided to initiate the next generation of SiLA. A result of this work is the SiLA 2 specification. Good ideas from SiLA 1.x were evolved. Many concepts and ideas were adopted from proven and emerging (Internet) standards, the Internet of Things (IoT) and web technologies.

The aim of this specification is to apply existing technologies and standards to the laboratory domain.

Therefore, this specification is designed to be “lean and mean” and simple. It only describes what is absolutely needed in the domain of lab automation, and otherwise references other open standards and protocols.

SiLA 2 has been designed such that it will not require proprietary libraries to be implemented. Long-term stability, wide support and acceptance have been critical criteria for the selection of the underlying standards and technologies.

The ultimate goal of SiLA is to enable a new kind of lab automation. To this end, SiLA 2 is designed on solid principles to enable true plug-and-play operation.

How to Get Started

This section provides pointers to the document sections relevant to specific stakeholders in the laboratory automation domain.

See also https://gitlab.com/SiLA2/sila_base/wikis/home.

Device Vendor, Service Provider, Device Designer

Assume you are the one that designs or provides a laboratory device or service. Part A helps to understand the layout of the SiLA 2 framework. The section Architecture is describing the main components, i.e. the ·SiLA Server·, ·SiLA Client· and ·Feature·. Once that is clear, one can start to design ·Features· in a manner without using too many technical details.

Software Developer

Assuming that you are the one that should implement a given specification, Part A is about the tech-agnostic declaration. Part B contains the mapping to the actual technical details.

SiLA Integrator

The main task of integrating systems into the lab is bringing ·SiLA Servers· and ·SiLA Clients· together. Since the technical back layer is nearly symmetrical, your approach will be very similar to the software developer. Skim the architecture of Part A and get familiar with the ·Feature Framework·. Technical details can be found in Part B.

Scientist / Lab Technician

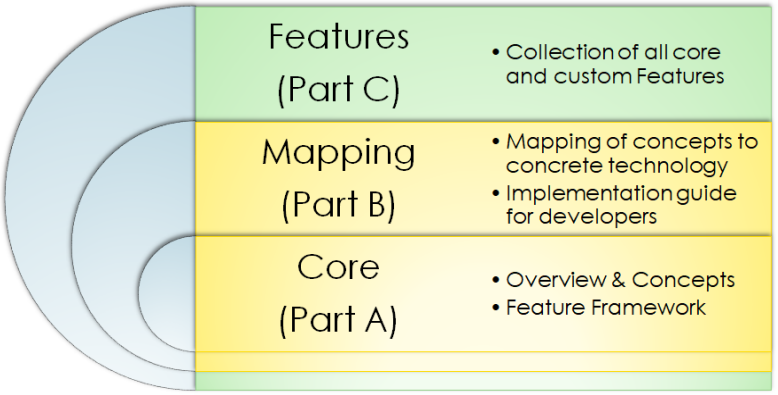
Since there should be no developing included in the ambit of running a laboratory, Part A, especially the SiLA 2 ·Feature Framework·, should be enough to understand it. In addition, since SiLA 2 is aiming to create a plug & play lab-environment, an interesting section might also be ·SiLA Server Discovery· and ·SiLA Feature Discovery·.

High Level Overview

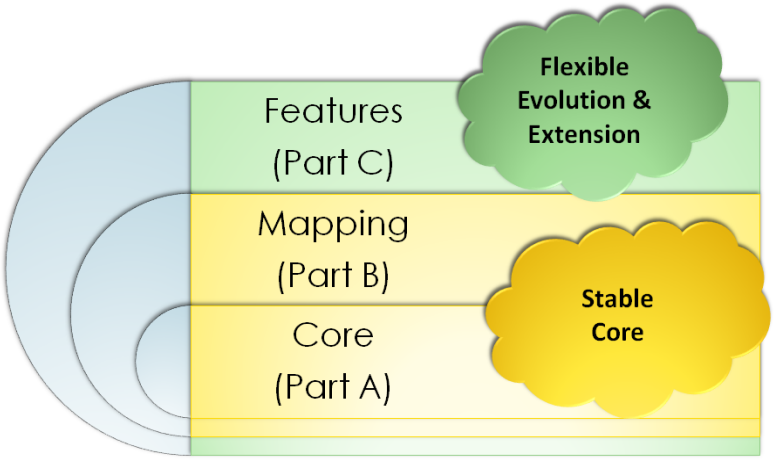
The SiLA 2 standardization process is a community process. The SiLA organization is investing a lot of effort into growing the community. Only a very active community that continues to develop the core standard, as well as ·Feature Definitions· on top will guarantee a long-term evolution and success of the standard.

Standardization Documents and Process

In order to make the SiLA 2 standard as future-proof as possible, concepts (Part A) have been separated from technical implementation details (Part B). The SiLA 2 standard is documented in several parts. The reason for this is that the individual parts are maintained and balloted separately.



The core and mapping specifications are written and maintained by the SiLA 2 Working group and kept as stable as possible. The flexible evolution and (vendor) specific extensions only happen on a ·Feature· level. This is reflected in our documentation layout:



Architectural Overview

The basic entities in SiLA 2 are the ·SiLA Client· and the ·SiLA Server·. A ·Connection· is opened either by the ·SiLA Client· (·Client-Initiated Connection Method·) or the ·SiLA Server· (·Server-Initiated Connection Method·). A ·Connection· is always established over a TCP/IP network (like an intranet or the Internet).

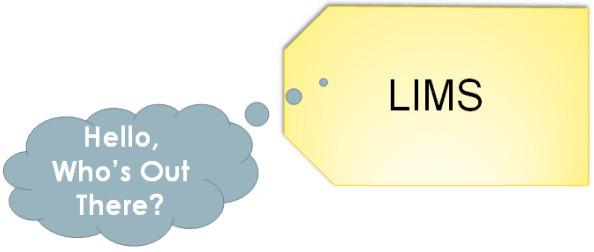
However, please note that ·SiLA Client· and ·SiLA Server· are only “roles” and not necessarily physical entities. A hardware device or a software application can in principle assume either role, even at the same time.

A central part of SiLA 2 is the discovery of ·SiLA Servers· by ·SiLA Clients·. ·SiLA Servers· advertise themselves so ·SiLA Clients· can find them in local networks.

A ·SiLA Server· is comparable to a “web server”, that is offering services to a web browser (the ·SiLA Client·). The behavior of a web server and web browser is comparable to a ·SiLA Server· and a ·SiLA Client· to a high extent.

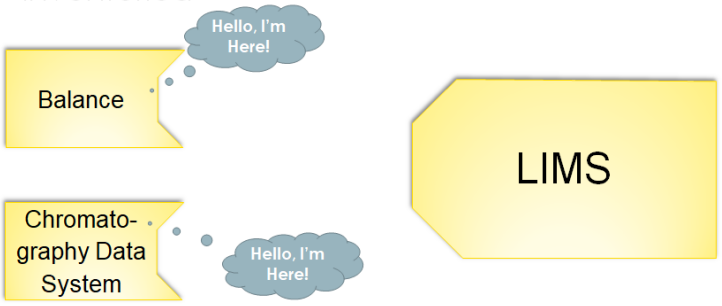
A ·SiLA Client· is looking for ·SiLA Servers·:

- LIMS Performs **SiLA Discovery**



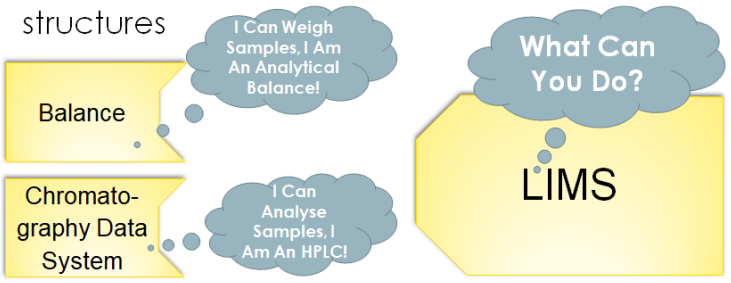
·SiLA Servers· may respond with the required information so the ·SiLA Client· knows how to connect. This includes all relevant information needed for connecting like the IP address and TCP port.

- Instruments And Services **Respond** And Are Inventoried



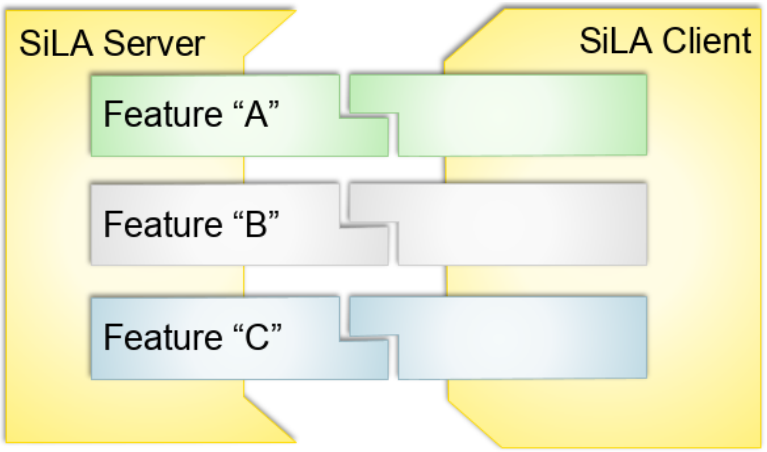
Once discovered, a ·SiLA Server· is able to advertise its capabilities. In SiLA 2, these capabilities are named ·Features·:

- Instruments and Services describe their **Features** (→ i.e. capabilities)
- **LIMS learns** how to communicate & data structures



One of the biggest assets of SiLA is, that the ·Feature· design will directly influence the user experience. This enables a whole new range of use cases in the smart laboratory.

A ·SiLA Server· may expose multiple ·Features·, with some ·Features· being able to modify the behavior of other ·Features·. For example, a ·SiLA Server· exposing a “Weighing Service” ·Feature· might also expose an “Authorization Service” ·Feature·. In turn, the ·Commands· offered by the “Weighing” ·Feature· are only accessible once authorized (through the “Authorization” ·Feature·).



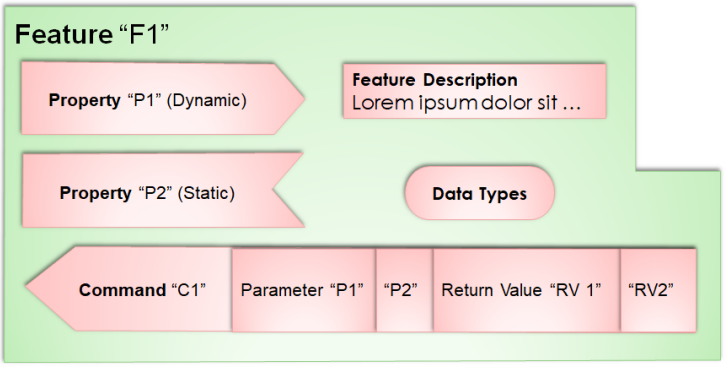
·Features· can also be seen as the application programming interface (API) definition of a ·SiLA Server·. ·Features· can define ·Properties·, ·Commands· and ·SiLA Data Types·. They also include a description of how to use them and any other information relevant to the execution or maintaining of the ·Feature·.

In contrast to an API in other technologies or standard formats, a SiLA ·Feature· can be designed by a subject matter expert (SME) who is not an IT expert. The “language” used to define a ·Feature· is close to the natural human language and therefore easily understandable for non-IT audiences.

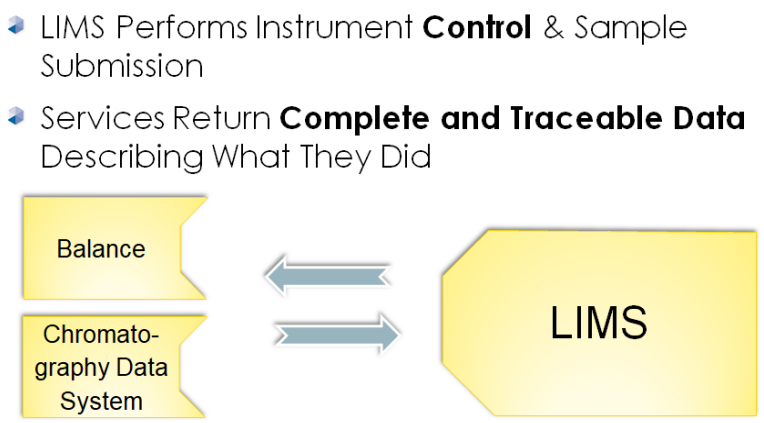
SiLA ·Features· and ·SiLA Feature Discovery· empower programmers to write software that is able to deal with any SiLA-compliant device, without knowing it before. Such ·SiLA Clients· can discover ·SiLA Servers· and their ·Features· and even operate them.

There are ·Features· that are provided with the SiLA 2 standard. Most of these ·Features· are “device agnostic” and can be used universally, independent of the nature of the ·SiLA Server·. But also private or vendor-specific ·Features· are possible. ·Features· can be added to the ·Online Feature Repository·, but SiLA 2 also works without a repository.

The SiLA Organization strives to find consensus for different ·Features· covering the same functionality and try to standardize one version of such ·Features·. However, ·Features· covering the same or similar functionality may also coexist.

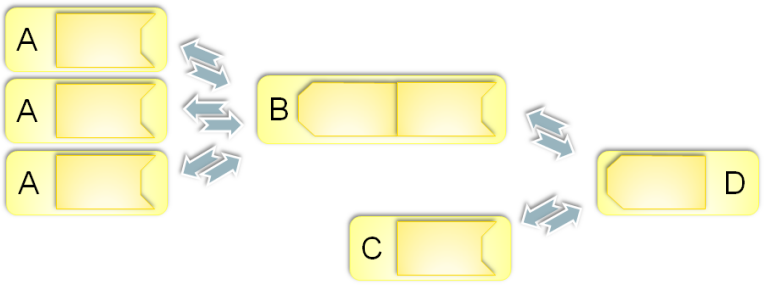


Once a ·SiLA Client· knows the ·Features· offered by a ·SiLA Server·, the ·SiLA Client· can start using these ·Features·:



SiLA does not specify or enforce any additional architectural constraints. An application architect or designer is free in how to setup ·SiLA Clients· and ·SiLA Servers·. However, the SiLA 2 Specification proposes best practices in many areas and aspects that should be followed to be SiLA-compliant. They are meant as good examples to encourage the design of ·Features· in the recommended way, but without the constraint of having to do so.

SiLA enables true plug & play and service orientation:



Each yellow box represents a ·SiLA Server·, ·SiLA Client· or both. The arrows indicate a ·Connection· between the ·SiLA Client· and the ·SiLA Server·: This illustrates a number of communication scenarios.

A. Laboratory devices, such as pipettors, plate handlers, plate sealers, i.e. ·SiLA Servers· implementing their respective ·Features·.
B. A process management system (PMS) orchestrating these laboratory devices, acting as a ·SiLA Client· to them and exposing a higher level ·Feature· as a ·SiLA Server·.
C. Another laboratory device, for example an analytical instrument offering a respective ·Feature· as a ·SiLA Server·.
D. A graphical user interface, acting as a ·SiLA Client· and controlling the PMS and the analytical device from a user's perspective.

SiLA Client and Server Interaction in a Nutshell

1. Establishing the ·Connection· Two ways of establishing a ·Connection· do exist:
    a. ·Client-Initiated Connection Method·: A ·SiLA Client· needs to know the ·Address· of the ·SiLA Server· to connect to it. A ·SiLA Client· can use ·SiLA Server Discovery· to determine the ·Address·. A ·SiLA Client· connects to the ·SiLA Server· at the ·Address·.

   b. ·Server-Initiated Connection Method·: A ·SiLA Server· needs to know the ·Address· of the ·SiLA Client· to connect to it. A ·SiLA Server· connects to the ·SiLA Client· at the ·Address·.
2. The SiLA Client retrieves the ·SiLA Service Feature· to get details about the ·SiLA Server· and discover ·Features· (·SiLA Feature Discovery·).
3. Ways of binding
   a. Late binding at run-time: a ·SiLA Client· discovers and uses ·Features· at run-time.
   b. Early binding at design time: the ·Features· a ·SiLA Client· is accessing were already known while designing the ·SiLA Client·.
4. ·SiLA Client· and ·SiLA Server· interaction:
   a. Executing an ·Unobservable Command·
   b. Executing an ·Observable Command·
   c. Reading a ·Property·
   d. Subscribing to a ·Property·
5. Closing the ·Connection·

Design Principles

- The SiLA 2 architecture is based on some fundamental aspects, like being explicit in everything and being strict in all specifications.
- Type safety is important.
- Clean separation of concerns (such as definition of the data and the transport of the data between involved parties). This has been realized through a clear separation between the core specification, ·Feature Definitions· and the mapping specification that provides the details for actually implementing SiLA.
- SiLA 2 is based on existing standards for interoperability of systems (such as Internet standards) and only defines those things (such as taxonomies / naming conventions) needed for the laboratory domain on top.
- A message-driven architecture that gracefully handles network interruptions.
- SiLA 2 should be accessible to a wide range of users. From software programmers to scientists and subject matter experts, SiLA 2 should be easy to understand.
- SiLA 2 is not a static specification. To remain relevant, it is continuously maintained and enhanced. Therefore, an important goal was to create working processes and ways to achieve consensus that facilitate (and do not hinder) evolution.
- Prepare the basis for a living ecosystem that helps evolving and improving the SiLA 2 Specification and fosters implementations.

Reference Implementations and Utilities

Reference implementations and utilities of the SiLA 2 working group can be found on https://gitlab.com/SiLA2. Please note, that these repositories are strongly supported by the SiLA organization but, except for https://gitlab.com/SiLA2/sila_base, the repositories are not a part of the official SiLA 2 Standard.

= = = START OF NORMATIVE PART = = =

Structure of the SiLA 2 Specification

[COMPLETE; as of 0.1]

The SiLA 2 Specification is a multi part specification:

- Part (A) - Overview, Concepts and Core Specification (current version of **this document**): contains the user requirements specification of SiLA 2. It describes **what** SiLA would like to achieve.
  It describes the core of SiLA 2 including the Features Framework in detail, but does not map to a specific implementation. This document deals with:
  - Overview of the design goals
  - SiLA 2 ·Features· specification
  - SiLA 2 ·Features· design rules
  - SiLA 2 ·Features· development and balloting process
  - Error handling and ·SiLA Data Types·
  - Security and Authentication
  - ·SiLA Server Discovery· and ·SiLA Feature Discovery·
- Part (B) - Mapping Specification (current version): describes **how** the user requirements shall be implemented. The mapping specification document describes the specific mapping to a technology and an actual implementation.
- Part (C) - Standard Features Index (current version): The Standard Features Index document is an index to ·Features· that are either standardized or currently being discussed to become standardized.

Terminology and Conformance Language

[COMPLETE; as of 0.1]

Unless otherwise noted, the entire text of this specification is normative. Exceptions include:

- Notes
- Sections explicitly marked non-normative
- Examples and their commentary
- Informal descriptions of details formally and normatively stated elsewhere (such informal descriptions are typically introduced by phrases like "Informally, ..." or "It is a consequence of ... that ...")

Explicit statements that some material is normative are not implying that other material is non-normative, other than the items mentioned in the list just described.

Special terms are defined at their point of introduction in the text.

For example:

[Definition: Term] a **Term** is something used with a special meaning. The definition is labeled as such and the **Term** it defines is displayed in boldface. The end of the definition is not specially marked in the displayed or printed text. Uses of defined **Terms** are links to their definitions, set off with middle dots, for instance ·Term·.

Normative text describes one or both of the following kinds of elements:

- Vital elements of the specification
- Elements that contain the conformance language keywords as defined by RFC2119 "Key words for use in RFCs to Indicate Requirement Levels"

Informative text is potentially helpful to the user, but dispensable. Informative text can be changed, added, or deleted editorially without negatively affecting the implementation of the specification. Informative text does not contain conformance keywords.

All text in this document between “START OF NORMATIVE PART” and “END OF NORMATIVE PART” is, by default, normative.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC2119 "Key words for use in RFCs to Indicate Requirement Levels".

Architecture & Important Definitions

[COMPLETE; as of 0.1]

SiLA’s architecture is built around the following elements:

SiLA Server

[COMPLETE; as of 0.1, updates: 1.1]

·SiLA Server·

[Definition: SiLA Server] A **SiLA Server** is a system (a software system, a laboratory instrument, or device) that offers ·Features· to a ·SiLA Client·. Every **SiLA Server** MUST implement the ·SiLA Service Feature·.

A ·SiLA Server· can either be a physical laboratory instrument (i.e. a spectrophotometer, a balance, a pH meter, ...) or a software system (i.e. a software system such as a Laboratory Information Management System - LIMS, a Laboratory Notebook - ELN, a Laboratory Execution System - LES, an Enterprise Resource Planning System - ERP, ...) that offers functionalities to a ·SiLA Client·. A ·SiLA Server· can offer a set of functionalities. All functionalities are specified and described by ·Features·.

A ·SiLA Server· MUST allow at least one ·Connection· at the same time. It is RECOMMENDED that a ·SiLA Server· allows parallel ·Connections· of multiple ·SiLA Clients· as its resources allow.

The ·SiLA Server· MAY decide to only allow one ·SiLA Client· to execute ·Commands· or access ·Properties· at the same time. However, it is RECOMMENDED that the ·SiLA Server· allows many ·SiLA Clients· to execute ·Commands· or access ·Properties· simultaneously.

A ·SiLA Server· MAY support ·Command· queuing and / or parallel ·Command· execution by one or multiple ·SiLA Clients·.

[Definition: Lifetime of a SiLA Server] The **Lifetime of a SiLA Server** is the time span between the state of power-up and being ready to accept ·Connections· and the shutdown process of a ·SiLA Server·, after which no new ·Connections· will be accepted.

SiLA Server Properties

[COMPLETE; as of 0.1]

A ·SiLA Server· has several properties used for its identification, see also The SiLA Service Feature:

SILA Server Name

[COMPLETE; as of 0.1]

Item	Description
SILA Server Name	[Definition: SiLA Server Name] The **SiLA Server Name** is a human readable name of the ·SiLA Server·. By default this name SHOULD be equal to the ·SiLA Server Type·. This property MUST be configurable via the ·SiLA Service Feature·'s "Set Server Name" Command. This property has no uniqueness guarantee. A **SiLA Server Name** is the ·Display Name· of a ·SiLA Server· (i.e. MUST comply with the rules for any ·Display Name·, hence be a string of UNICODE characters of maximum 255 characters in length).

SiLA Server Type

[COMPLETE; as of 0.1]

Item	Description
SiLA Server Type	[Definition: SiLA Server Type] The **SiLA Server Type** is a human readable ·Identifier· of the ·SiLA Server· used to describe the entity that the ·SiLA Server· represents. For example, the make and model for a hardware device (·SiLA Device·). A **SiLA Server Type** MUST comply with the rules for any ·Identifier· and start with an upper-case letter (A-Z) and MAY be continued by lower and upper-case letters (A-Z and a-z) and digits (0-9) up to a maximum of 255 characters in length.

SiLA Server UUID

[COMPLETE; as of 0.1]

Item	Description
SiLA Server UUID	[Definition: SiLA Server UUID] The **SiLA Server UUID** is a ·UUID· of a ·SiLA Server·. Each ·SiLA Server· MUST generate a ·UUID· once, to uniquely identify itself. It needs to remain the same even after the ·Lifetime of a SiLA Server· has ended.

SiLA Server Version

[COMPLETE; as of 0.1]

Item	Description
SiLA Server Version	[Definition: SiLA Server Version] The **SiLA Server Version** is the version of the SiLA Server. A "Major" and a "Minor" version number (e.g. 1.0) MUST be provided, a Patch version number MAY be provided. Optionally, an arbitrary text, separated by an underscore MAY be appended, e.g. “3.19.373_mighty_lab_devices”.

SiLA Server Vendor URL

[COMPLETE; as of 0.1]

Item	Description
SiLA Server Vendor URL	[Definition: SiLA Server Vendor URL] The **SiLA Server Vendor URL** is the URL to the website of the vendor or the website of the product of this SiLA Server. This URL SHOULD be accessible at all times. The URL is a Uniform Resource Locator as defined in RFC 1738.

SiLA Server Description

[COMPLETE; as of 0.1]

Item	Description
SiLA Server Description	[Definition: SiLA Server Description] The **SiLA Server Description** is the description of the SiLA Server. It SHOULD include the use and purpose of this SiLA Server.

SiLA Device

[COMPLETE; as of 0.1]

[Definition: SiLA Device] A **SiLA Device** is a ·SiLA Server· that is a physical thing made or adapted for a particular purpose, especially a piece of mechanical or electronic equipment. A **SiLA Device** is a specialization of a ·SiLA Server· with additional requirements regarding joining a communication network.

A ·SiLA Device· MUST support automatic connection to a communication network.

A ·SiLA Device· MUST support automatic determination of its ·Address· in the communication network.

SiLA Client

[COMPLETE; as of 0.1]

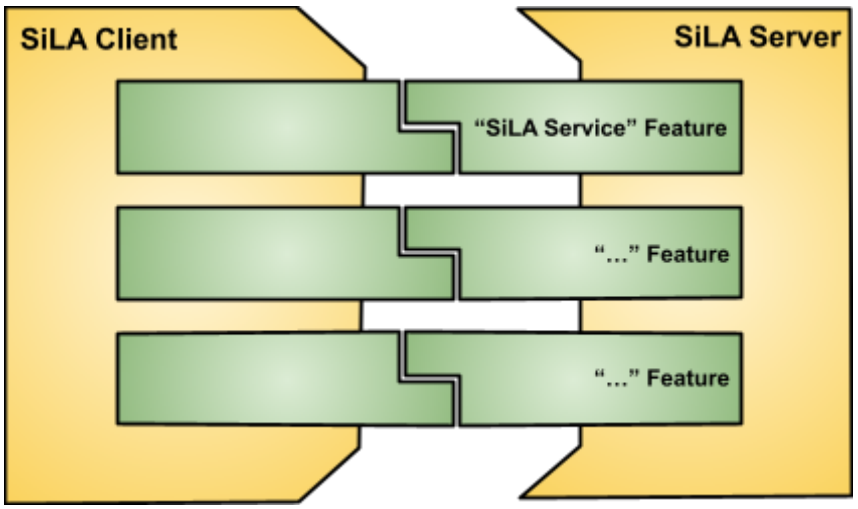
·SiLA Client·

[Definition: SiLA Client] A **SiLA Client** is a system (a software system, a laboratory instrument, or device), that is using ·Features· offered by a ·SiLA Server·.

Feature

[COMPLETE; as of 0.1]

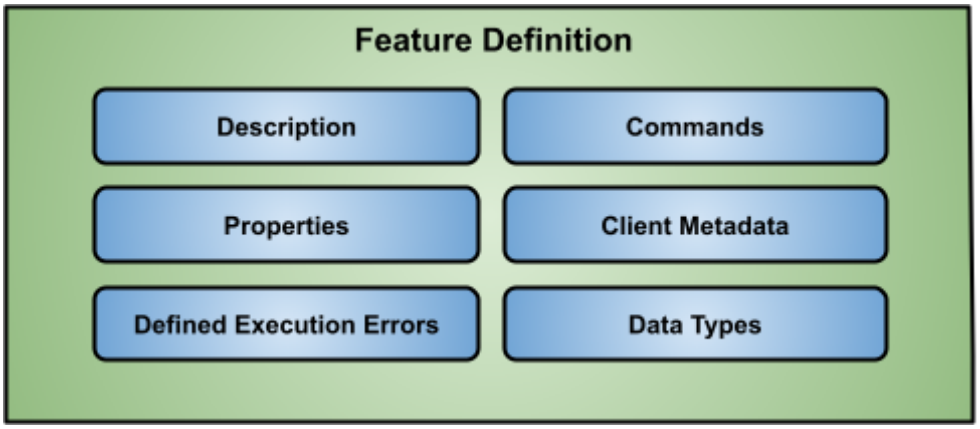
[Definition: Feature] Each **Feature** describes a specific behavior of a ·SiLA Server· (e.g. the ability to measure a spectrum, to register a sample in a LIMS, control heating, etc.). **Features** are implemented by a ·SiLA Server· and used by a ·SiLA Client·. The ·SiLA Service Feature· MUST be implemented by each ·SiLA Server·. The ·SiLA Service Feature· offers basic information about the ·SiLA Server· and about all other **Features** the ·SiLA Server· implements.



·SiLA Client· connecting to ·SiLA Server· through ·Features·

The number and nature of the ·Features· that a ·SiLA Server· implements MUST be static for the whole ·Lifetime of a SiLA Server· (no dynamic adding or removing of ·Features· is allowed).

·Features· are specified within a ·Feature Definition·. SiLA defines a set of best practices on how ·Features· should be designed.



High Level overview of a ·Feature·

Whereas everybody can freely design ·Features·, a goal of the SiLA organization is as well to standardize certain ·Features· that can be commonly used. Therefore, SiLA 2 also defines a process for maintaining ·Features· on a best practice basis.

Connection

[COMPLETE; as of 0.1, updates: 1.1]

[Definition: Connection] A **Connection** is the communication channel between a ·SiLA Client· and a ·SiLA Server·, established over a communication network. All information exchange between a ·SiLA Client· and a ·SiLA Server· MUST be exchanged through the **Connection**.

[Definition: Address] An **Address** is an identifier that uniquely identifies a ·SiLA Client· or a ·SiLA Server· in a communication network.

·SiLA Client· connected to ·SiLA Server·

Connection Method

[COMPLETE; as of 0.1, updates: 1.1]

[Definition: Connection Method] The **Connection Method** specifies which of the parties is establishing the ·Connection·.

Client-Initiated Connection Method

[COMPLETE; as of 1.1]

[Definition: Client-Initiated Connection Method] With the **Client-Initiated Connection Method**, the ·SiLA Client· is establishing the ·Connection· to the ·SiLA Server·. This ·Connection Method· is available as of ·SiLA 2 Version· “0.1”.

The ·Client-Initiated Connection Method· MUST be supported always by all ·SiLA Servers·.

Server-Initiated Connection Method

[COMPLETE; as of 1.1]

[Definition: Server-Initiated Connection Method] With the **Server-Initiated Connection Method**, (a.k.a. “cloud connectivity” or “reverse connection”), the ·SiLA Server· is establishing the ·Connection· to the ·SiLA Client·. This ·Connection Method· is available as of ·SiLA 2 Version· “1.1”.

A ·SiLA Server· conforming to ·SiLA 2 Version· equal to or larger than “1.1” SHALL support the ·Server-Initiated Connection Method·. Such a ·SiLA Server· indicates the availability of the ·Server-Initiated Connection Method· support by offering the ·Connection Configuration Service Feature·.

Establishing a Connection

[COMPLETE; as of 0.1, updates: 1.1]

The ·Connection· between a ·SiLA Client· and ·SiLA Server· can either be established by the ·SiLA Client· (“client-initiated” ·Connection·) or the ·SiLA Server· (“server-initiated” ·Connection·). Independent of which side opened the ·Connection·, all mechanisms specified in this document MUST work in the same way.

The ·Connection· MAY be closed by the ·SiLA Client· or the ·SiLA Server· at any time. It is RECOMMENDED that the ·SiLA Server· only closes the ·Connection· in exceptional cases, e.g. due to a lack of resources or when the ·Lifetime of a SiLA Server· is reached.

Both ·SiLA Server· and ·SiLA Client· MUST handle closing or losing of the ·Connection· gracefully, at anytime during the ·Lifetime of a SiLA Server·.

In case a ·Connection· has been closed or lost, the ·SiLA Server· MUST stop sending ·Observable Property· changes or ·Command Execution Info· events from ·Observable Command· execution to the affected ·SiLA Client·. The ·SiLA Server· MUST, however, continue processing ·Observable Commands· that the ·SiLA Client· has initiated and that were already accepted for ·Command· execution by the ·SiLA Server·. ·Command Execution UUIDs· MUST remain valid as specified, so that a ·SiLA Client· is still able to use it after having re-connected to the ·SiLA Server·.

See also ·Connection Error·.

SiLA Client Request

[COMPLETE; as of 0.1]

[Definition: SiLA Client Request] A **SiLA Client Request** is a piece of information sent from a ·SiLA Client· to a ·SiLA Server· within a ·Connection·.

A ·SiLA Client Request· within a ·Connection·

Content of a SiLA Client Request

[COMPLETE; as of 0.1]

A ·SiLA Client Request· is a composition of a ·Header· followed by a ·Payload·.

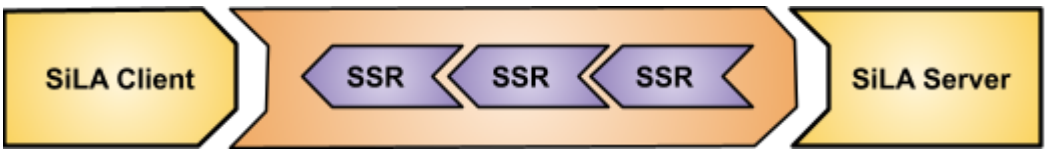
Components of a ·SiLA Client Request·

SiLA Server Response

[COMPLETE; as of 0.1]

[Definition: SiLA Server Response] A **SiLA Server Response** is a piece of information sent from a ·SiLA Server· to a ·SiLA Client· within a ·Connection·. A **SiLA Server Response** is always returned in reply to a ·SiLA Client Request·. For subscriptions (to ·Observable Properties· or ·Observable Commands·), a streamed **SiLA Server Response** is sent as long as the subscription is active.

A ·SiLA Server Response· within a ·Connection·



A streamed ·SiLA Server Response· within a ·Connection·

Content of a SiLA Server Response

[COMPLETE; as of 0.1]

A ·SiLA Server Response· is a composition of a ·Header· followed by a ·Payload·.

Components of a ·SiLA Server Response·

A streamed ·SiLA Server Response· is a composition of a ·Header· followed by any number of ·Payloads·, followed by a ·Trailer·.

Components of a streamed ·SiLA Server Response·

Header and Trailer

[COMPLETE; as of 0.1]

[Definition: Header] A **Header** is the first part of a ·SiLA Client Request· or ·SiLA Server Response·. A **Header** contains for example ·SiLA Client Metadata· of the ·Payload· and error information.

[Definition: Trailer] A **Trailer** is the last part of a streamed ·SiLA Server Response·. A **Trailer** contains for example error information.

Payload

[COMPLETE; as of 0.1]

[Definition: Payload] The **Payload** is the actual data exchanged between a ·SiLA Client· and a ·SiLA Server·.

Feature Framework

[COMPLETE; as of 0.1]

[Definition: Feature Framework] **Feature Framework** is an overarching ·Term· describing the intention of ·Features· in SiLA 2, how to design, store and maintain them.

Each ·SiLA Server· offers a certain capability and each ·SiLA Client· wants to use or access this capability. The capability or behavior is exposed through one or more ·Features·.

·Features· are a key component of the SiLA 2 Standard as they define the interaction between a ·SiLA Client· and a ·SiLA Server·. Every single ·Feature· describes a certain aspect of the overall behavior of the ·SiLA Server·. Collectively the ·Features· create the ·SiLA Server· behavior.

Every ·Feature· MUST be created in such a way that it can function alone. It is paramount that the ·Feature· is easy to understand only by reading the ·Feature Definition·.

Note that the behavior of the ·Features· can be altered by other ·Features· thus changing the overall behavior of the ·SiLA Server· when implemented (for example: AuthorizationService).

Each ·Feature· MUST encapsulate complexity as much as possible. The ·Feature Framework· is designed in a way that non-technical experts, such as scientists or subject matter experts (SMEs) are able to create a ·Feature Definition·. However and in addition, the ·Feature Framework· is designed in a way that ·Feature Definitions· are easily interpretable and implementable by technical people.

It is RECOMMENDED that a ·Feature· does have little or no state (that is it SHALL follow a stateless design). However, a ·Feature Designer· MAY decide that a stateful design is necessary and MAY decide to expose status through SiLA ·Properties· or ·Command Responses· of ·Commands·.

Only one mandatory ·Feature·, the ·SiLA Service Feature·, MUST be implemented by each and every ·SiLA Server·.

A ·SiLA Server· MAY expose one or more ·Feature Version·(s) of the same ·Feature· at the same time.

A ·SiLA Client· SHALL be able to use one or more ·Feature Version·(s) of the same ·Feature· simultaneously.

SiLA 2 not only specifies how to design ·Features· but also defines the process of how ·Features· are to be maintained through ·Attributes·.

Designing Features

[COMPLETE; as of 0.1]

[Definition: Feature Designer] A **Feature Designer** is a person designing a ·Feature·. Usually this SHALL be a subject matter expert for the domain that the ·Feature· addresses.

·Features· are the central point in the SiLA Standard definition. Therefore correct conception and implementation of the ·Features· is paramount. For this to be assured a ·Feature Designer· MUST adhere to the ·Feature· design rules and conventions as described in the chapters below.

·Features· MUST be designed without a specific implementation in mind as much as possible. The focus should be on describing the behavior (behavioral design), clarity and reusability.

All names and descriptions indicated to be human readable (generally all content of a ·Feature Definition· that is marked to be human readable) MUST be in the American English language. A ·SiLA Server· implementer MAY decide to provide localized versions, see also Internationalization.

Standard Features

[COMPLETE; as of 0.1]

[Definition: SiLA Standard Feature] A **SiLA Standard Feature** is a ·Feature· that has been standardized by the SiLA organization according to the standardization procedure. An officially standardized and released ·Feature· MUST have the ·Originator· set to “org.silastandard” and the ·Maturity Level· to “Normative”. There is a formal process for becoming a **SiLA Standard Feature** according to the SiLA by-laws.

Storing Features

[COMPLETE; as of 0.1]

All ·Feature Definitions· with ·Originator· “org.silastandard” MUST be stored in the ·Online Feature Repository·, all other ·Feature Definitions· MAY also be stored there.

[Definition: Online Feature Repository] The **Online Feature Repository** is a place for storing ·Feature Definitions·. It can be found on GitLab.

·Feature Definitions· MUST be stored as an XML file conforming to the ·Feature Definition Language· at the GitLab repository sila_base under the following path:

“ feature_definitions / {·Originator·} / {·Category·} / {·Feature Identifier·} ‘-v’ {Major ·Feature Version·} ‘_’ {Minor ·Feature Version·} ‘.sila.xml’ ”

where each dot (“.”) separated component of the ·Originator· and each dot (“.”) separated component of the ·Category· will be a new subfolder in this path structure.

Examples

[COMPLETE; as of 0.1]

sila_base / feature_definitions / org / silastandard / core / SiLAService-v1_0.sila.xml

sila_base / feature_definitions / org / silastandard / examples / GreetingProvider-v1_0.sila.xml

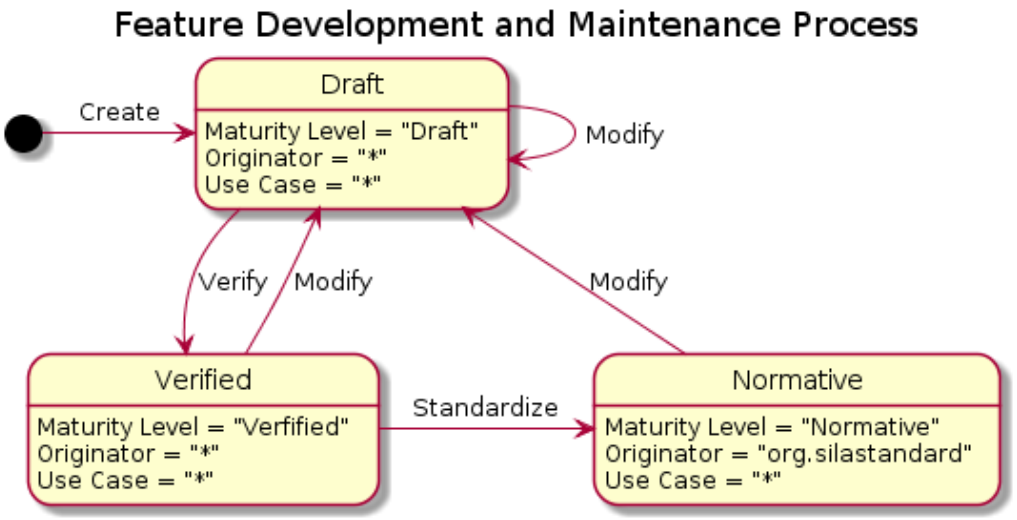
Maintaining Features

[COMPLETE; as of 0.1]

Each ·Feature· MUST have an ·Originator·, ·Category·, ·Maturity Level· and ·Feature Version· assigned to it, see Feature Attributes. Each ·Feature· has a life cycle and needs to be managed

throughout its whole life. SiLA defines a life cycle for ·Features·, its states and possible transitions between these states. The possible states of a ·Feature· are defined by its ·Attributes·.

The state diagram below illustrates how these states are related to each other.



The following transitions in the life cycle of a ·Feature Definitions· are valid:

Creating Features

[COMPLETE; as of 0.1]

Initial creation of a new ·Feature· SHALL result in a ·Feature· with ·Maturity Level· ·Draft·.

Verifying Features

[COMPLETE; as of 0.1]

After verification that a ·Feature· meets the SiLA standard definitions and best practices, a ·Feature· SHALL have ·Maturity Level· ·Verified·.

As of now, only members of the SiLA 2 working group are allowed to officially verify that a ·Feature· meets the SiLA standard definitions and best practices. Any ·Feature· to be verified MUST be uploaded to the ·Online Feature Repository· and a request for official verification has to be sent to a member of the SiLA 2 working group to start the process. This process might be changed in the future to allow officially accredited bodies to perform the verification.

Modifying Features

[COMPLETE; as of 0.1]

Any modification of a ·Feature· or its ·Attributes· (except for modifying the ·Maturity Level·, which is implicit) SHALL end up in the state ·Draft·. During this transition, the ·Feature Version· ·Attribute· MUST be updated as specified in section Feature Version below. ·Feature Definitions· MUST be maintained in a way to enable full backwards and forwards compatibility.

Standardizing Features

[COMPLETE; as of 0.1]

The standardization of a ·Feature· follows a process specified in the by-laws of SiLA. Only ·Features· with ·Originator· "org.silastandard" SHALL be standardizable and able to reach the ·Maturity Level· ·Normative·.

Feature Definition

[COMPLETE; as of 0.1]

[Definition: Feature Definition] The **Feature Definition** describes a certain behavior of a ·Feature· in an exact and very specific way.

Designing good ·Features· is an art and this specification provides examples on how to design good ·Features· in the best practice sections located in each section.

A ·Feature Definition· contains the following information:

- Details about the ·Feature·, such as ·Feature Identifier·, ·Feature Display Name· and ·Feature Description·
- Details about all ·Commands· offered
- Details about all ·Properties· offered
- A Definition of all ·SiLA Data Types· used
- A Definition of all SiLA Client Metadata used
- Details about all ·Execution Errors· that can happen while executing ·Commands·, accessing Properties or using SiLA Client Metadata
- Additional ·Attributes· of the ·Feature·

Best Practice

[COMPLETE; as of 0.1]

The ·Feature Definition· MUST specify the behavior it is modeling in a self-contained and complete way. It MUST contain an extensive ·Feature Description· of the behavior it models. ·Constraints·, like under which preconditions a ·Command· should be called, valid ·Parameter· ranges and any other dependencies MUST be detailed. In case the behavior that the ·Feature· is describing is exporting a state, all states and their transitions MUST be described.

Feature Identifier

[COMPLETE; as of 0.1]

Item	Description
Feature Identifier	[Definition: Feature Identifier] A **Feature Identifier** is the ·Identifier· of a ·Feature·. Each ·Feature· MUST have a **Feature Identifier**. All ·Features· sharing the scope of the same ·Originator· and ·Category· MUST have unique **Feature Identifiers.** Uniqueness MUST be checked without taking lower and upper case into account, see Uniqueness of Identifiers.

Best Practice

[COMPLETE; as of 0.1]

The ·Feature Identifier· SHOULD be in CamelCase (i.e. written in the upper camel case style).

The ·Feature Identifier· SHOULD be in the form of one or more noun(s) with one of the three following suffixes: Service, Provider or Controller, whereas the suffix SHOULD be selected as follows:

- Controller: “active” ·Features·; if a ·Feature· is responsible for the management of a certain task.
- Provider: “passive” ·Features·; these ·Features· are responsible for gathering information and making it available to be called upon.
- Service: All ·Features· that do not fall in one of these above mentioned categories, or fall in both.

This form of nomenclature creates a name that describes what the ·Feature· is, while also giving insight into the behavior it describes.

Examples

[COMPLETE; as of 0.1]

SiLAService, InitializationController, LockController, PauseController, SimulationController, LocalizationProvider.

Feature Display Name

[COMPLETE; as of 0.1]

Item	Description
Feature Display Name	[Definition: Feature Display Name] A **Feature Display Name** is the ·Display Name· of a ·Feature·.

Best Practice

[COMPLETE; as of 0.1]

It is RECOMMENDED to use the ·Feature Identifier· with spaces between the words to make it more readable. The name SHOULD indicate what capabilities the ·Features· describes. The ·Feature Display Name· MUST be human readable text in American English (see also Internationalization).

While the ·Feature· Designer is free to choose whatever name they want, just taking the ·Feature Identifier· is good practice since the ·Feature Identifier· SHOULD already entail all the information that the ·Feature Display Name· should display. Giving a short but concise description of the functionality of the ·Feature·. It is important to SiLA that the environment is easy to understand. A human readable name that describes the ·Feature· gives even non-developers a brief description of that ·Feature· and what a user can expect from it.

Example

[COMPLETE; as of 0.1]

The ·SiLA Service Feature· (·Identifier· “SiLAService”) has a ·Feature Display Name· of “SiLA Service”, likewise “InitializationController” becomes “Initialization Controller” and “LockController” “Lock Controller”.

Feature Description

[COMPLETE; as of 0.1]

Item	Description
Feature Description	[Definition: Feature Description] A **Feature Description** is the ·Description· of a ·Feature·. A **Feature Description** MUST describe the behaviors / capabilities the ·Feature· models in human readable form and with as many details as possible. The **Feature Description** SHOULD contain all details about the ·Feature· as described under best practice below. The **Feature Description** MUST be human readable text in American English (see also Internationalization).

Best Practice

[COMPLETE; as of 0.1]

The most important part of a good ·Feature· is the ·Feature Description·. The ·Feature Description· MUST be a plain text, detailing:

- The behavior of the ·Feature·.
- The intention of the ·Feature· and the context when it is applicable.
- Whether the involved ·Commands· have to be called in a specific sequence or whether there are specific pre-conditions.
- Whether there are dependencies between ·Commands· and which these are.
- How this ·Feature· modifies the behavior of other ·Features·, if applicable
- A “How to use”-guide for the ·Feature·.

It is important to note that the ·Feature Description· is not the same thing as ·Feature Definition· as the latter is the whole ·Feature· while the ·Feature Description· is only the descriptional text elucidated above. Since easy comprehension and implementation is one of the main goals of SiLA, the ·Feature Description· serves the expressive service of clarifying any issues in implementation or use of the ·Feature·. A good ·Feature· will have a ·Feature Description· that provides answers to the users questions in regards to the use of the ·Feature·.

Example

[COMPLETE; as of 0.1]

For example the ·SiLA Service Feature· has the following ·Feature Description·:

> The Feature each SiLA Server MUST implement. Any interaction described in this feature MUST not affect the behavior of any other Feature. It specifies Commands and Properties to discover the Features a SiLA Server implements as well as details about the SiLA Server.

Commands

[COMPLETE; as of 0.1, updates: 1.1]

[Definition: Command] A **Command** models an action that will be performed on a ·SiLA Server·. A **Command** MAY except ·Command Parameters· and MAY return ·Command Responses· after **Command** execution or MAY provide ·Intermediate Command Responses· during execution. There are ·Unobservable Commands· and ·Observable Commands·.

Item	Description
Command Identifier	[Definition: Command Identifier] A **Command Identifier** is the ·Identifier· of a ·Command·. A **Command Identifier** MUST be unique within the scope of a ·Feature·. Uniqueness MUST be checked without taking lower and upper case into account, see Uniqueness of Identifiers.
Command Display Name	[Definition: Command Display Name] A **Command Display Name** is the human readable ·Display Name· of a ·Command·.
Command Description	[Definition: Command Description] A **Command Description** is the ·Description· of a ·Command·.

Best Practice

[COMPLETE; as of 0.1]

A ·Feature Designer· SHOULD adhere to the following guidelines:

Item	Guideline
Command Identifier	The ·Command Identifier· SHOULD be in CamelCase (i.e. written in the upper camel case style). The ·Command Identifier· is RECOMMENDED to be of the form: **Verb[Adjective(s)][Noun(s)]** (verb in imperative singular, optional adjective(s), optional noun(s)). The verb is to indicate the action to be taken, and the adjective/noun the rest of the description. The developers and/or users can then easily understand or identify which ·Feature· does what. Thus making the correct use of that ·Feature· easier.
Command Display Name	A human readable name for this ·Command·. It is RECOMMENDED to use the ·Command Identifier· with spaces between the words. The utility and functionality of the ·Command· SHOULD be indicated by the ·Command Display Name·.
Command Description	Details and any additional information that is necessary for the complete understanding of the use of this ·Command· SHOULD also be added to the ·Command Description·. This information could contain pre- or post conditions or ·Parameter· constraints and dependencies on status.

Example

[COMPLETE; as of 0.1]

GetFeatureDefinition

Command Identifier	GetFeatureDefinition
Command Display Name	Get Feature Definition
Command Description	Get all details on one ·Feature· through the fully qualified ·Feature Identifier·.
Command Observable Setting	No

Observable and Unobservable Commands

[COMPLETE; as of 0.1]

A ·Feature Designer· MUST specify how a ·Command· will have to behave when executed. ·Commands· can either be ·Observable Commands· or ·Unobservable Commands·, as described below.

[Definition: Unobservable Command] Any ·Command· for which observing the progress or status of the ·Command· execution on the ·SiLA Server· is not possible or does not make sense, SHOULD be defined as an **Unobservable Command**, i.e. a ·Command· with item “Observable” set to “No”.

Note: When using an ·Unobservable Command· it is possible that the ·SiLA Client· is never able to receive the ·Command Response· (e.g. due to network failures). Use an ·Observable Command· avoid these cases.

[Definition: Observable Command] Any ·Command· for which observing the progress or status of the ·Command· execution on the ·SiLA Server· is possible and makes sense, e.g. measuring a spectrum, SHOULD be an **Observable Command**, i.e. a ·Command· with item “Observable” set to “Yes”.

Item	Description
Command Observable Setting	[Definition: Command Observable Setting] The **Command Observable Setting** specifies whether a ·Command· is an ·Observable Command· or an ·Unobservable Command·. The value MUST be either “Yes” (·Observable Command·) or “No” (·Unobservable Command·). For details, see Command Execution.

Best Practice

[COMPLETE; as of 0.1]

A ·Feature Designer· SHOULD decide whether a ·Command· is observable or not based on these best practices:

1. It is a potentially long lasting operation, for which a human being expects a non-immediate completion: The ·Command· should be observable.
2. The designer wants to make sure that the result is not lost, even if the connection is lost: The command should be observable.
3. It is only about sending data from a client to server, e.g. changing the settings: The ·Command· should be unobservable.

Command Parameters

[COMPLETE; as of 0.1]

[Definition: Command Parameter] A **Command Parameter** is a ·Parameter· of a ·Command·. It MUST be submitted with the ·Command· execution request. A ·Command· MAY have one or more **Command Parameters**.

Item	Description
Command Parameter Identifier	[Definition: Command Parameter Identifier] The **Command Parameter Identifier** is the ·Identifier· of a ·Command Parameter·. A **Command Parameter Identifier** MUST be unique within the scope of all ·Parameters· of a ·Command·.
Command Parameter Display Name	[Definition: Command Parameter Display Name] A **Command Parameter Display Name** is the ·Display Name· of a ·Command Parameter·.
Command Parameter Description	[Definition: Command Parameter Description] A **Command Parameter Description** is the ·Description· of a ·Command Parameter·.
Command Parameter Data Type	[Definition: Command Parameter Data Type] A **Command Parameter Data Type** is The ·SiLA Data Type· of a ·Command Parameter·.

Best Practice

[COMPLETE; as of 0.1]

A ·Feature Designer· SHOULD adhere to the following guidelines:

Item	Guidelines
Command Parameter Identifier	The ·Command Parameter Identifier· SHOULD be in CamelCase (i.e. written in the upper camel case style). The ·Command Parameter Identifier· SHOULD be of the form: **AdjectiveNoun(s)** (optional adjective, singular or plural noun(s), depending on context). The adjective/noun used SHOULD indicate the utility and functionality of the ·Command Parameter·. In other words, it SHOULD describe what it is that is used by the ·Command·.
Command Parameter Display Name	A human readable name for this ·Command Parameter·. It is RECOMMENDED to use the ·Command Parameter Identifier· with spaces between the words.

Command Parameter Description	The ·Description· of this ·Command Parameter· SHOULD give an accurate depiction of what the ·Command Parameter· is and what it is used for by the ·Command·. In case it is necessary for other ·Commands· or has certain prerequisites this SHOULD also be included in the ·Command Parameter Description·. This is to further reduce the amount of errors and facilitate the use of each of the ·Commands· simply by reading the ·Command Parameter Description·.

Examples

[COMPLETE; as of 0.1]

TargetTemperature

Command Parameter Identifier	TargetTemperature
Command Parameter Display Name	Target Temperature
Command Parameter Description	The target temperature that the server will try to reach. Note that the ·Command· might be completed at a temperature that it evaluates to be close enough.
Command Parameter Data Type	·SiLA Real Type· with a ·Unit Constraint· for °C.

Command Responses

[COMPLETE; as of 0.1]

[Definition: Command Response] A **Command Response** contains a result of a ·Command· execution. A ·Command· MAY have one or more **Command Responses**.

In contrast to many programming languages that only offer one **Command Response**, SiLA 2 likes to use symmetries whenever possible, hence the decision to have the possibility to have zero or more **Command Responses** like it is the case for ·Parameters·.

Item	Description
Command Response Identifier	[Definition: Command Response Identifier] A **Command Response Identifier** is the ·Identifier· of a ·Command Response·. A **Command Response Identifier** MUST be unique within the scope of al ·Command Responses· of a ·Command·. Uniqueness MUST be checked without taking lower and upper case into account, see Uniqueness of Identifiers.
Command Response Display Name	[Definition: Command Response Display Name] A **Command Response Display Name** is the ·Display Name· of a ·Command Response·.

Command Response Description	[Definition: Command Response Description] A **Command Response Description** is a ·Description· of a ·Command Response·.
Command Response Data Type	[Definition: Command Response Data Type] A **Command Response Data Type** is the ·SiLA Data Type· of a ·Command Response·.

Best Practice

[COMPLETE; as of 0.1]

A ·Feature Designer· SHOULD adhere to the following guidelines:

Item	Guidelines
Command Response Identifier	The ·Identifier· of a ·Command Response· SHOULD be in CamelCase (i.e. written in the upper camel case style). The ·Command Response Identifier· SHOULD be in the form: **AdjectiveNoun(s)** (optional adjective, singular or plural noun(s), depending on context). The adjective/noun used SHOULD indicate the utility and functionality of the ·Command Response·. In other words, it SHOULD describe what it is that is returned by the ·Command·.
Command Response Display Name	A human readable name for the ·Command Response·. It is RECOMMENDED to use the ·Command Response Identifier· with spaces between separate words.
Command Response Description	The ·Description· of the ·Command Response· SHOULD give an accurate depiction of what it is and what the ·Command· returns.

A ·Feature Designer· SHOULD NOT use a ·Command Response· to inform about a successful or failed completion of the ·Command· (e.g. do not specify a ·Command Response· of a ·SiLA Boolean Type· with ·Command Response Identifier· “Success”). It is RECOMMENDED to use an ·Execution Error· to communicate an error and a successful completion is given by no error being thrown.

Examples

[COMPLETE; as of 0.1]

Command Response Identifier	Barcode
Command Response Display Name	Barcode
Command Response Description	The barcode related to the sample produced in this ·Command·. The barcode is of type Code128.

Command Response Data Type	·SiLA String Type· (Potentially with an appropriate ·Constraint·.).

Intermediate Command Responses

[COMPLETE; as of 0.1]

[Definition: Intermediate Command Response] An **Intermediate Command Response** is a partial ·SiLA Server Response· of an ·Observable Command·. An ·Observable Command· MAY have one or more **Intermediate Command Responses**.

·Unobservable Commands· MUST NOT have ·Intermediate Command Responses·.

Item	Description
Intermediate Command Response Identifier	[Definition: Intermediate Command Response Identifier] An **Intermediate Command Response Identifier** is the ·Identifier· of an ·Intermediate Command Response·. An **Intermediate Command Response Identifier** MUST be unique within the scope of all ·Intermediate Command Responses· of a ·Command·. Uniqueness MUST be checked without taking lower and upper case into account, see Uniqueness of Identifiers.
Intermediate Command Response Display Name	[Definition: Intermediate Command Response Display Name] An **Intermediate Command Response Display Name** is the ·Display Name·of an ·Intermediate Command Response·.
Intermediate Command Response Description	[Definition: Intermediate Command Response Description] An **Intermediate Command Response Description** is the ·Description· of an ·Intermediate Command Response·.
Intermediate Command Response Data Type	[Definition: Intermediate Command Response Data Type] An **Intermediate Command Response Data Type** is the ·SiLA Data Type· of an ·Intermediate Command Response·.

Best Practice

[COMPLETE; as of 0.1]

It is a good practice to specify ·Intermediate Command Responses· for long lasting ·Commands· that are able to produce intermediate results during the ·Command· execution. Since ·Intermediate Command Responses· are subscription-based, there can be no guarantee that they are actually received. So they SHOULD not be used for information that is expected to be transmitted. Instead, ·Command Responses· SHOULD be used for such information.

A ·Feature Designer· SHOULD adhere to the following guidelines:

Item	Guidelines
Intermediate Command Response Identifier	The ·Intermediate Command Response Identifier· SHOULD be in CamelCase (i.e. written in the upper camel case style). The ·Intermediate Command Response Identifier· SHOULD be in the form: **AdjectiveNoun(s)** (optional adjective, singular or plural noun(s), depending on context).

	The adjective/noun used SHOULD indicate the utility and functionality of the ·Intermediate Command Response·. In other words, it SHOULD describe what it is that is returned by the ·Command·.
Intermediate Command Response Display Name	A human readable name for the ·Intermediate Command Response·. It is RECOMMENDED to use the ·Intermediate Command Response Identifier· with spaces between separate words.
Intermediate Command Response Description	The ·Description· of the ·Intermediate Command Response· SHOULD give an accurate depiction of what it is and what the ·Command· returns as an intermediate result.

Examples

[COMPLETE; as of 0.1]

Intermediate Command Response Identifier	IntermediateAbsorbanceSpectrum
Intermediate Command Response Display Name	Intermediate Absorbance Spectrum
Intermediate Command Response Description	The intermediate spectrum as measured during the execution of this command, if available, as described in the description of the command.
Intermediate Command Response Data Type	·SiLA Binary Type· with a ·Content Type Constraint·.

Defined Execution Errors in Command Execution Context

[COMPLETE; as of 0.2]

One or more ·Defined Execution Errors· MAY be specified for a ·Command·. The ·Defined Execution Errors· themselves are defined in the context of the ·Feature· (and not the ·Command·), so they can be re-used within the ·Feature·, e.g. for different ·Commands·.

Item	Description
Defined Execution Errors	A list of ·Defined Execution Error Identifiers· referring to all the ·Defined Execution Errors· that can happen when executing this ·Command·.

Command Execution

[COMPLETE; as of 0.1]

A ·SiLA Server· MUST only start a ·Command· execution after all ·Parameters· have been properly received (i.e. there was no ·Connection Error· while sending the ·Parameters·) and after validating all ·Parameters· (i.e. there was no ·Validation Error·).

In case one or more ·SiLA Clients· try to execute the identical or different ·Commands· at the same time, the ·SiLA Server· MUST decide whether:

- to execute the ·Commands· in parallel,
- to execute the ·Commands· sequentially, by queueing ·Commands· or
- to reject the ·Command· execution.

Whether or not a ·SiLA Server· supports parallel ·Command· execution, queueing of ·Command· execution or whether a ·SiLA Server· rejects multiple, simultaneous ·Command· executions is upon the sole discretion of the ·SiLA Server· implementation.

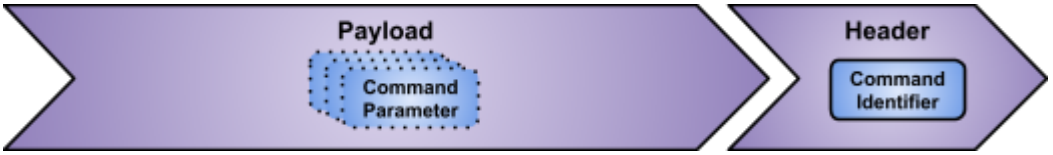
In case the ·SiLA Server· does not allow the ·Command· execution, a ·Command Execution Not Accepted Error· (·Framework Error·) MUST be issued by the ·SiLA Server·. A ·SiLA Server· MAY for instance decline executing a ·Command· if another ·Command· is already running and the ·SiLA Server· does not support parallel ·Command· execution or queueing of ·Commands·.

The ·SiLA Server· MUST always accept ·Command· execution for ·Commands· of the ·SiLA Service Feature·.

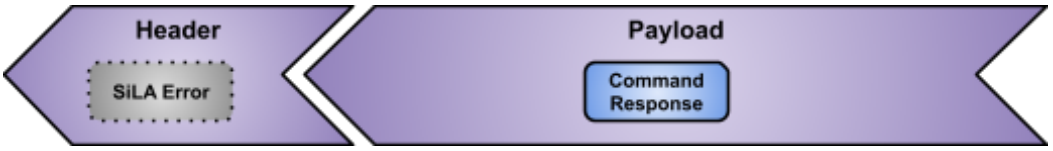
Unobservable Command

[COMPLETE; as of 0.1]

A ·SiLA Server· SHOULD execute ·Unobservable Commands· as quickly as possible, as there is no opportunity for a ·SiLA Client· to see the current progress or remaining time of the execution.

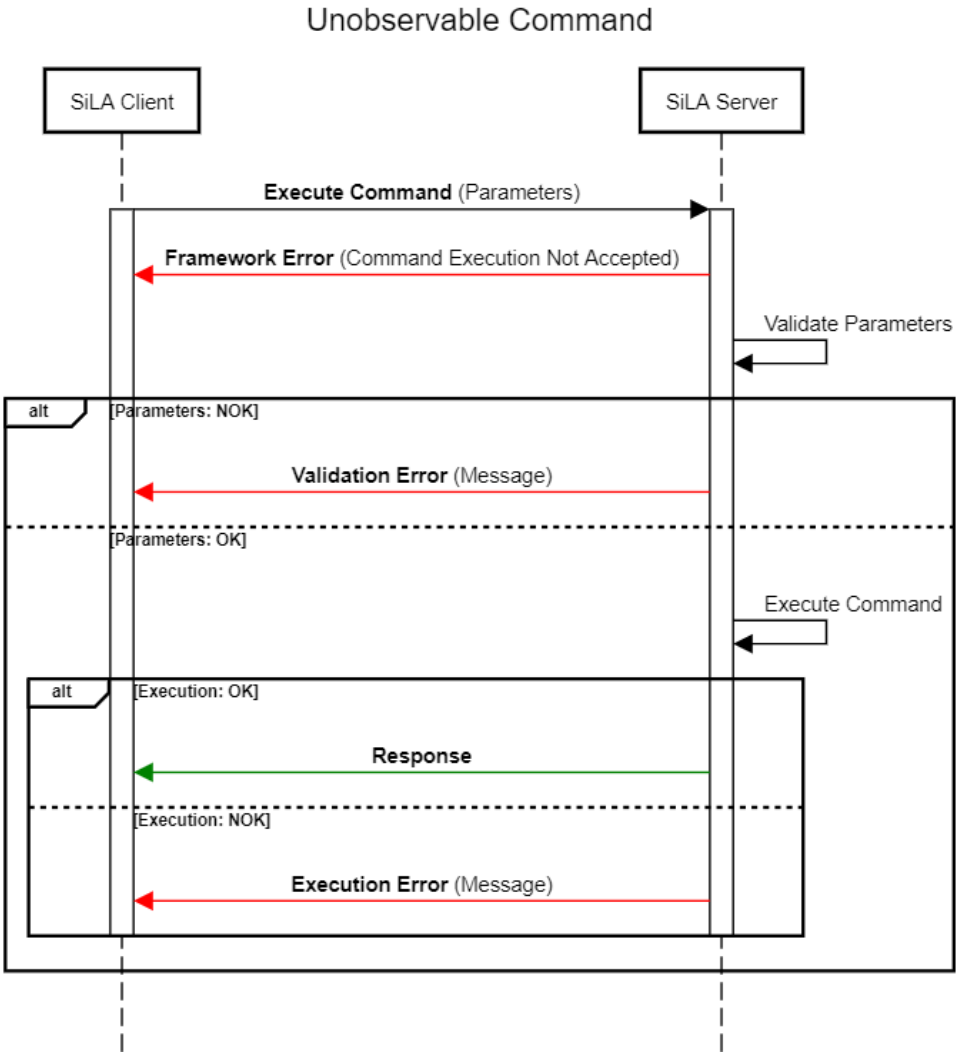


Structure of a ·SiLA Client Request· to initiate an ·Unobservable Command· execution



Structure of a ·SiLA Server Response· in reply to an ·Unobservable Command· execution

Sequence diagram of an ·Unobservable Command· execution:



Observable Command

[COMPLETE; as of 0.1]

After ·Command· initiation and accepting the execution, the ·SiLA Server· MUST return the ·Command Execution UUID· to the ·SiLA Client·.

The ·SiLA Server· MAY optionally also return a ·Lifetime of Execution· to the ·SiLA Client·. If no ·Lifetime of Execution· is provided, then the ·Command Execution UUID· MUST be valid as long as the ·Lifetime of a SiLA Server·. It is RECOMMENDED to return a defined ·Lifetime of Execution· as otherwise the ·SiLA Server· doesn't have an ability to control the associated resources.

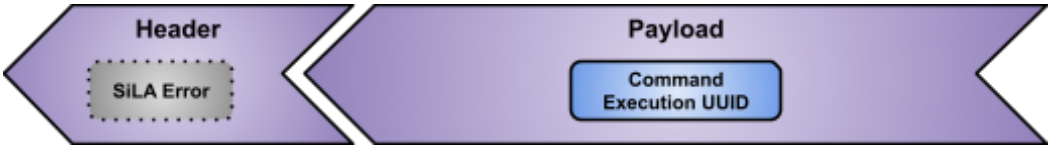
Afterwards, the ·SiLA Server· MUST either start the ·Command· execution, schedule it for later ·Command· execution or issue an error, at its sole discretion and depending on its capabilities.

[Definition: Command Execution UUID] A **Command Execution UUID** is the ·UUID· of a ·Command· execution. It is unique within one instance of a ·SiLA Server· and its lifetime (·Lifetime of a SiLA Server·).

[Definition: Lifetime of Execution] The **Lifetime of Execution** is the duration during which a ·Command Execution UUID· is valid. The **Lifetime of Execution** is always a relative duration with respect to the point in time the ·SiLA Server· initiated the response to the ·SiLA Client· (the point in time when the SiLA Server returns the ·Command Execution UUID· to the ·SiLA Client·). It is the

responsibility of the ·SiLA Client· to account for potential transmission delays between ·SiLA Server· and ·SiLA Client·.

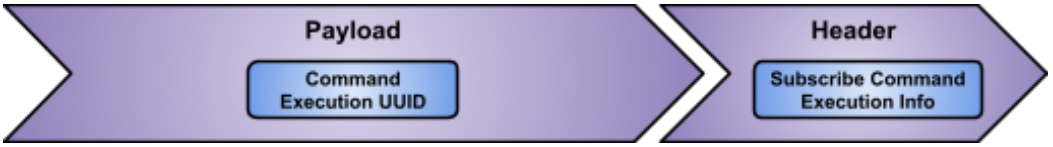
Structure of a ·SiLA Client Request· to initiate an ·Observable Command· execution



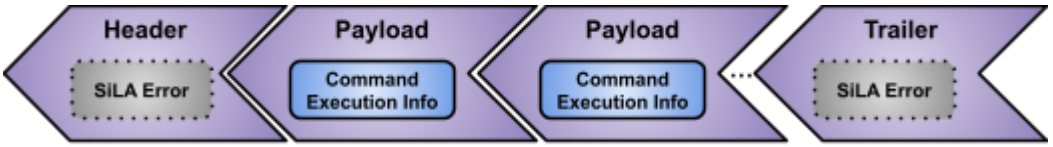
Structure of a ·SiLA Server Response· in reply to an ·Observable Command· initiation

The ·SiLA Server· MUST provide ·Command Execution Info· on request or via subscription.

[Definition: Command Execution Info] The **Command Execution Info** provides information about the current status of a ·Command· being executed. It consists of the ·Command Execution Status·, and optionally the ·Progress Info· and an ·Estimated Remaining Time·. In addition, an updated ·Lifetime of Execution· MUST be provided, if a ·Lifetime of Execution· has been provided at ·Command· initiation.



Structure of a ·SiLA Client Request· to subscribe to the ·Command Execution Info·



Structure of a streamed ·SiLA Server Response· in reply to a subscription to the ·Command Execution Info·

[Definition: Command Execution Status] The **Command Execution Status** provides details about the execution status of a ·Command·. It is either, and in this sequence, first “Command Waiting”, second “Command Running” and third “Command Finished Successfully” or “Command Finished With Error”. The **Command Execution Status** cannot be reverted back to a previous state that has already been left. That is, once the ·Command· is running, the state cannot go back to waiting etc.

[Definition: Progress Info] **Progress Info** is the estimated progress of a ·Command· execution, in percent (0...100%).

[Definition: Estimated Remaining Time] **Estimated Remaining Time** is the estimated remaining execution time of a ·Command·.

If the ·SiLA Server· provides an updated ·Lifetime of Execution· as part of the ·Command Execution Info·, the updated lifetime MUST always result in an absolute lifetime that is equal to or

greater than prior lifetimes reported. The absolute lifetime of a ·Command Execution UUID· MUST never be reduced, but MAY be extended any time.

Note that it follows that, if the ·SiLA Server· did not provide a ·Lifetime of Execution· in the ·Command· initiation, it is illegal to provide an updated ·Lifetime of Execution· in the ·Command Execution Info·. Vice versa, if the ·SiLA Server· did provide a ·Lifetime of Execution·, the ·SiLA Server· MUST always send updated ·Lifetime of Execution· in the ·Command Execution Info·.

In order to retrieve or subscribe to the ·Command Execution Info·, ·Command Response· or ·Intermediate Command Responses·, the ·SiLA Client· MUST use the ·Command Execution UUID·.

Retrieving or subscribing to ·Intermediate Command Responses· is only available if ·Intermediate Command Responses· are specified in the ·Feature· for the corresponding ·Command·.

A ·SiLA Client· MAY share the ·Command Execution UUID· with other ·SiLA Clients· to allow them to access the same information about the ·Observable Command· identified by the shared ·Command Execution UUID·.

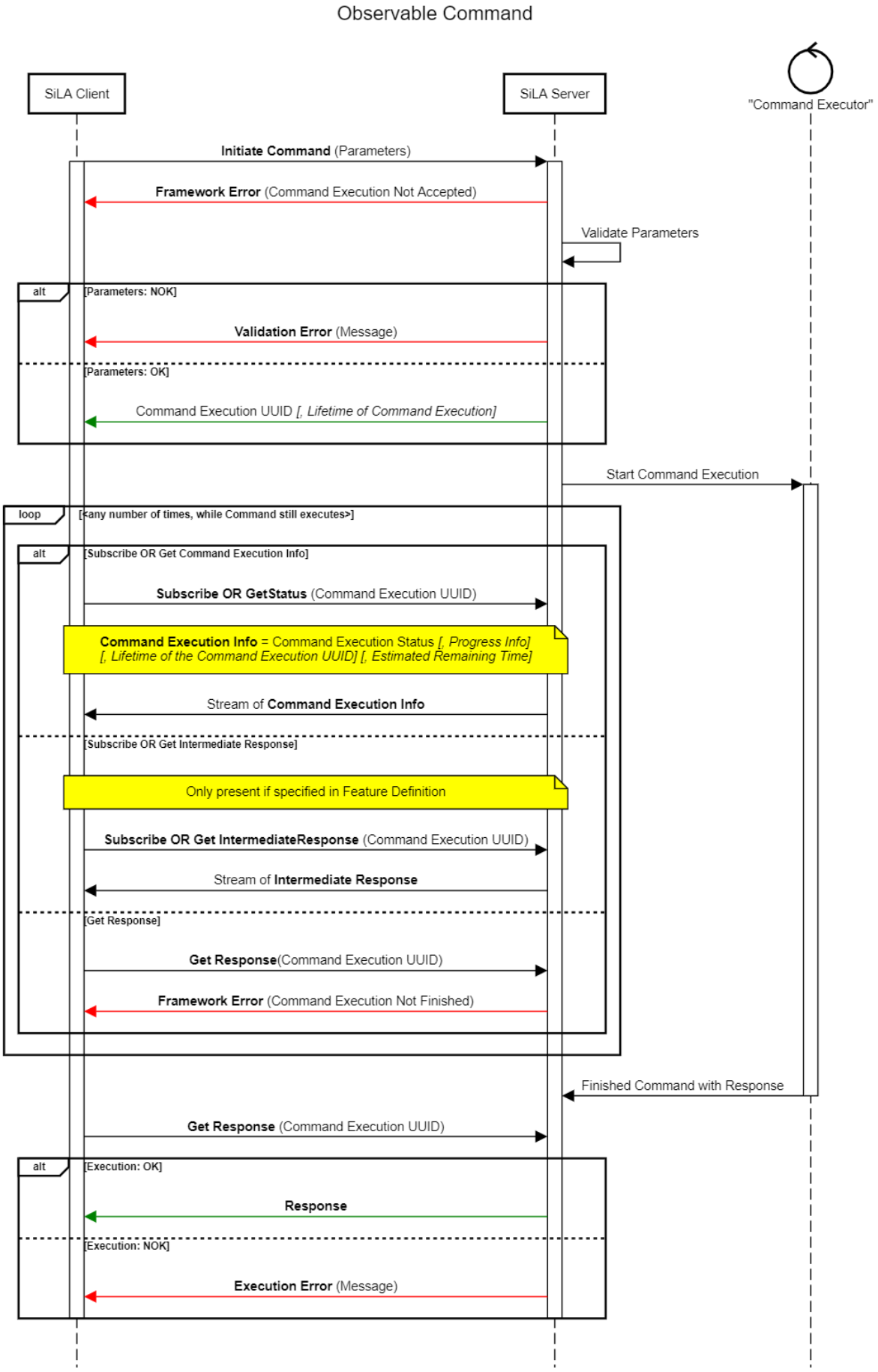
If a ·SiLA Client· uses an invalid ·Command Execution UUID· for querying a ·SiLA Server·, the ·SiLA Server· MUST return an ·Invalid Command Execution UUID Error·.

After ·Command· execution (status: Command Finished Successfully or Command Finished With Error), the ·SiLA Client· MAY ask the ·SiLA Server· for the ·Command Response· for the respective ·Command Execution UUID·. The ·SiLA Server· MUST either return the ·Command Response· or an error if the ·Command· execution was not successful.

If a ·SiLA Client· asks for a ·Command Response· when the ·Command· execution has not finished yet (status: Command Waiting or Command Running), the ·SiLA Server· MUST return a ·Command Execution Not Finished Error·.

·Command Execution UUIDs· of an ·Observable Command· MUST remain valid for a duration that is defined by the ·Lifetime of Execution·. After the ·Command Execution UUID· has expired the ·Command Execution UUID· MAY be invalidated and all subscriptions associated with this ·Command Execution UUID· MAY be canceled. The ·SiLA Server· MAY then also delete the ·Command Response· and free all resources associated with it.

Sequence diagram of executing an ·Observable Command·:


Observable Command
SiLA Client
SiLA Server
"Command Executor"
Initiate Command (Parameters)
Framework Error (Command Execution Not Accepted)
Validate Parameters
alt
[Parameters: NOK]
Validation Error (Message)
[Parameters: OK]
Command Execution UUID [, Lifetime of Command Execution]
Start Command Execution
loop
[<any number of times, while Command still executes>]
alt
[Subscribe OR Get Command Execution Info]
Subscribe OR GetStatus (Command Execution UUID)
Command Execution Info = Command Execution Status [, Progress Info] [, Lifetime of the Command Execution UUID] [, Estimated Remaining Time]
Stream of Command Execution Info
[Subscribe OR Get Intermediate Response]
Only present if specified in Feature Definition
Subscribe OR Get IntermediateResponse (Command Execution UUID)
Stream of Intermediate Response
[Get Response]
Get Response(Command Execution UUID)
Framework Error (Command Execution Not Finished)
Finished Command with Response
Get Response (Command Execution UUID)
alt
[Execution: OK]
Response
[Execution: NOK]
Execution Error (Message)

Properties

[COMPLETE; as of 0.1, updates: 1.1]

[Definition: Property] A **Property** describes certain aspects of a ·SiLA Server· that do not require an action on the ·SiLA Server·. **Properties** may be changed by actions of the SiLA server. **Properties** are always read-only.

A ·Property· MUST be Observable or Unobservable. The two different types of ·Properties· SHALL be separated by their possible reading behaviors.

[Definition: Unobservable Property] An **Unobservable Property** is a ·Property· that can be read at any time but no subscription mechanism is provided to observe its changes.

[Definition: Observable Property] An **Observable Property** is a ·Property· that can be read at any time and that offers a subscription mechanism to observe any change of its value.

Reading a ·Property· MUST NOT have side effects on the ·SiLA Server·, i.e. the actual reading process itself SHALL NOT affect the value of any ·Property· of the ·SiLA Server·.

Item	Description
Property Identifier	[Definition: Property Identifier] A **Property Identifier** is the ·Identifier· of a ·Property·. A **Property Identifier** MUST be unique within the scope of a ·Feature·. Uniqueness MUST be checked without taking lower and upper case into account, see Uniqueness of Identifiers.
Property Display Name	[Definition: Property Display Name] A **Property Display Name** is the ·Display Name· of a ·Property·.
Property Description	[Definition: Property Description] A **Property Description** is the ·Description· of a ·Property·.
Property Observable Setting	[Definition: Property Observable Setting] The **Property Observable Setting** specifies whether a ·Property· is an ·Observable Property· or an ·Unobservable Property·. The value MUST be either “Yes” (·Observable Property·) or “No” (·Unobservable Property·). For details see Accessing Properties.
Property Data Type	[Definition: Property Data Type] A **Property Data Type** is the ·SiLA Data Type· of a ·Property·.

Best Practice

[COMPLETE; as of 0.1]

A ·Feature Designer· SHOULD adhere to the following guidelines:

Item	Guideline

Property Identifier	The ·Property Identifier· SHOULD be in CamelCase (i.e. written in the upper camel case style). The ·Property Identifier· SHOULD be of the form: Adjective[Noun(s)] (optional adjective, singular or plural noun(s), depending on context).
Property Display Name	It is RECOMMENDED to use the ·Property Identifier· with spaces between the words.
Property Description	The ·Description· of the ·Property· SHOULD give an accurate, in depth depiction of what the ·Property· is. If the ·Property· has a ·Command· that specifically can write to the ·Property· this SHOULD be described here. Any further details, necessary for the use of the ·Property· SHOULD be mentioned here.

·Properties· are always read-only. If a ·Feature Designer· likes to enable write access to a ·Property·, he or she has to explicitly define ·Command·(s) that directly (e.g. a “setter” ·Command·) or indirectly (e.g. a ·Command· that implicitly triggers the change of a ·Property·) change the value of a ·Property·.

The following reasons led to the decision not to allow writing of ·Property· values directly:

- SiLA would like to encourage stateless designs and designs that have the least possible side effects. Allowing writable ·Property· could lead to unwanted side effects.
- Writing ·Property· values could lead to ·Validation Errors·. Whereas it can be expected that a ·Command· results in an error, a user would not expect an error by just writing a new value to a ·Property·.

Observable or Unobservable Property? Design Considerations

Here are some guidelines that help ·Feature Designers· to decide whether to make a ·Property· an ·Observable Property· or ·Unobservable Property·:

As a general rule, ·Properties· should be made ·Observable Properties· if a client should be notified about any change of the ·Property· value.

However, there are reasons for making a ·Property· an ·Unobservable Property·, even if its value can change over time:

- The change of the ·Property· value is not very frequent and asking a ·SiLA Client· to poll for new values seem legitimate
- ·SiLA Clients· are most likely not interested in getting regular updates of the ·Property· value
- An ·Observable Property· adds effort during implementation and usage and a ·Feature Designer· may explicitly decide to avoid this additional effort to keep this effort low.

Note: The ·SiLA Server Name· of the ·SiLA Service Feature· was chosen to be an ·Unobservable Property· as the working group considered it to be important that the implementation of the ·SiLA Service Feature· remains as simple as possible.

Defined Execution Errors in Property Reading Context

[COMPLETE; as of 0.2]

One or more ·Defined Execution Errors· MAY be specified for a ·Property·. The ·Defined Execution Errors· themselves are defined in the context of the ·Feature· (and not the ·Property·), so they can be re-used within the ·Feature·, e.g. among different ·Properties·.

Item	Description
Defined Execution Errors	A list of ·Defined Execution Error Identifiers· referring to all the ·Defined Execution Errors· that can happen when accessing this ·Property·.

Accessing Properties

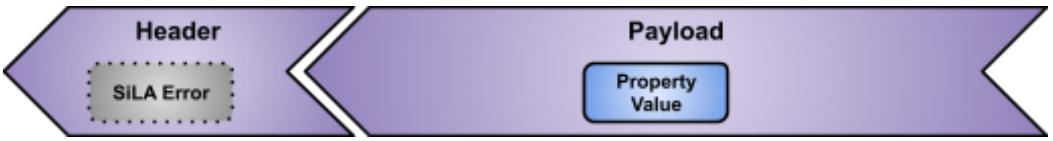
[COMPLETE; as of 0.1]

Reading a Property Value

[COMPLETE; as of 0.1]

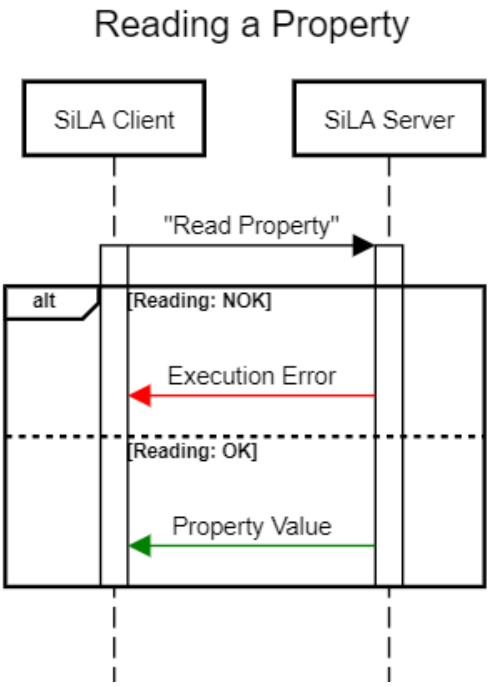
A ·SiLA Client· SHALL be able to read the value of an ·Observable Property· or an ·Unobservable Property· at any time within the ·Lifetime of a SiLA Server·. The ·SiLA Server· MUST return the ·Property·'s current value or issue an ·Execution Error· in case the ·SiLA Server· is unable to determine the value of the ·Property·.

Structure of a ·SiLA Client Request· to initiate a ·Property· read



Structure of a ·SiLA Server Response· in reply to a ·Property· read

Sequence diagram of reading an ·Observable Property· or an ·Unobservable Property·:



Subscribing to an Observable Property

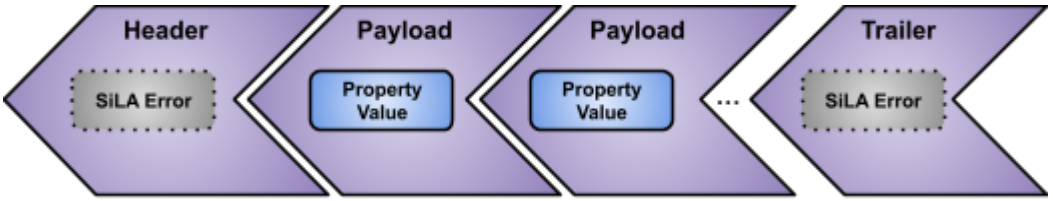
[COMPLETE; as of 0.1]

[Definition: Property Subscription] A **Property Subscription** is the subscription of an ·Observable Property· by a ·SiLA Client·.

The ·SiLA Server· MUST allow the ·SiLA Client· to subscribe to any ·Observable Property·. The ·SiLA Server· MUST return the ·Property·'s current value once when subscribed, as if the ·SiLA Client· was reading a ·Property· once. A new ·Property· value is sent whenever the value changes.

In case the ·SiLA Server· is unable to determine the current value of the ·Property·, the ·SiLA Server· MUST issue an ·Execution Error· and MUST cancel the subscription after that.

Structure of a ·SiLA Client Request· to initiate a ·Property Subscription·



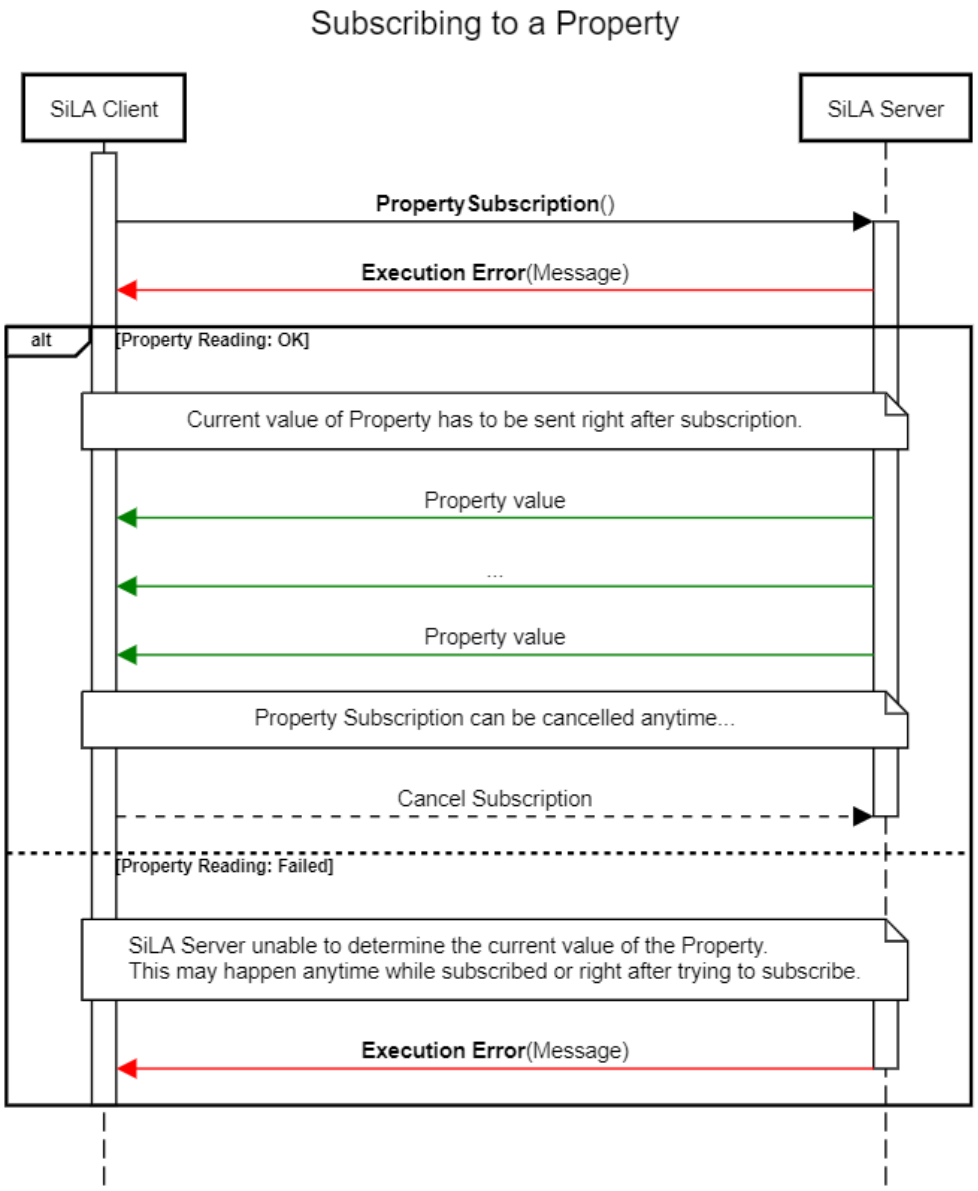
Structure of a streamed ·SiLA Server Response· in reply to a ·Property Subscription·

A ·Property Subscription· MAY be canceled by the ·SiLA Client· at any time.

A ·Property Subscription· SHALL implicitly be canceled by an interruption or loss of the ·Connection· (·Connection Error·) or by explicitly closing the ·Connection· by the ·SiLA Client·. The ·SiLA Server· SHALL detect these cases and stop sending new ·Property· values to ·SiLA Clients·

whose ·Connection· has been closed or lost. In these cases, it is the responsibility of the ·SiLA Client· to re-establish the ·Property Subscription·, if desired.

Sequence diagram of subscribing to an ·Observable Property·:



Examples

[COMPLETE; as of 0.1]

An example of an ·Observable Property· is e.g. a "Bath Temperature". An example of an ·Unobservable Property· is e.g. a “Serial Number”.

SiLA Client Metadata

[COMPLETE; as of 0.2]

A ·Feature Designer· MAY specify ·SiLA Client Metadata· that a ·SiLA Client· can send together with a ·Command· execution, reading or subscribing to a ·Property·.

[Definition: SiLA Client Metadata] **SiLA Client Metadata** is information that a ·SiLA Server· expects to receive from a ·SiLA Client· when executing a ·Command· or reading or subscribing to a ·Property·. If expected **SiLA Client Metadata** is not received, a Invalid Metadata Framework Error must be issued. This must be checked before parameter validation. Each **SiLA Client Metadata** has a specific ·Metadata Identifier· and a ·SiLA Data Type·. Metadata is intended for small pieces of data, and transmission might fail for values larger than 1 KB.

·SiLA Client Metadata· MUST NOT affect the ·SiLA Service Feature·. That means that every call of the ·SiLA Service Feature· MUST NOT contain any ·SiLA Client Metadata·. If a ·SiLA Server· receives ·SiLA Client Metadata· within the header of a ·SiLA Service Feature· call, it MUST issue a ·No Metadata Allowed Error·.

A ·SiLA Server· that implements a ·Feature· which defines ·SiLA Client Metadata· MUST provide a list for each ·SiLA Client Metadata· that contains all ·Features· / ·Commands· / ·Properties· that are affected by that ·SiLA Client Metadata·. The affected ·Features· / ·Commands· / ·Properties· are identified by their respective ·Fully Qualified Identifiers·, of which a ·Feature Identifier· represents all ·Commands· and ·Properties· of the respective ·Feature·. The list for all ·Features· / ·Commands· / ·Properties· affected by ·SiLA Client Metadata· MUST not change during the ·Lifetime of a SiLA Server·.

Note: As the ·SiLA Service Feature· MUST never be affected by ·SiLA Client Metadata·, the list of affected ·Features·, ·Commands· or ·Properties· MUST never contain any ·Commands· or ·Properties· of the ·SiLA Service Feature· or the ·SiLA Service Feature· ·Identifier· itself.

If a ·SiLA Server· implements a ·Feature· that defines ·SiLA Client Metadata·, the ·SiLA Server· MUST expect the ·SiLA Client Metadata· to be present as specified in the list of affected ·Features· / ·Commands· / ·Properties· or issue an ·Invalid Metadata Error· otherwise.

In case a ·SiLA Server· receives ·SiLA Client Metadata· unexpectedly (i.e. not in the list of affected ·Features·, ·Commands· or ·Properties·), the ·SiLA Server· MUST ignore it.

·SiLA Client Metadata·, if specified, MUST be sent to the ·SiLA Server· by the ·SiLA Client· when:

- Executing ·Unobservable Commands·
- Executing an ·Observable Command·; only with the ·Command· initiation (see ·Observable Command·, InitiateCommand; not with Subscribe- or GetCommandExecutionInfo, Subscribe- or GetIntermediateResponse or GetResponse)
- Reading a ·Property·
- Executing a ·Property Subscription·

·SiLA Client Metadata· is sent as part of the ·Header· of a ·SiLA Client Request·

Item	Description

Metadata Identifier	[Definition: Metadata Identifier] A **Metadata Identifier** is the ·Identifier· of a ·SiLA Client Metadata·. A **Metadata Identifier** MUST be unique within the scope of a ·Feature·. Uniqueness MUST be checked without taking lower and upper case into account, see Uniqueness of Identifiers.
Metadata Display Name	[Definition: Metadata Display Name] A **Metadata Display Name** is the ·Display Name· of a ·SiLA Client Metadata·.
Metadata Description	[Definition: Metadata Description] The **Metadata Description** is the ·Description· of a ·SiLA Client Metadata·.
Metadata Data Type	[Definition: Metadata Data Type] A **Metadata Data Type** is the ·SiLA Data Type· of a ·SiLA Client Metadata·.

Best Practice

[COMPLETE; as of 0.2]

Use ·SiLA Client Metadata· for concerns that affect more than one ·Feature· on a ·SiLA Server· at the same time. Some examples are given below.

Item	Description
Metadata Identifier	The ·Metadata Identifier· SHOULD be in CamelCase (i.e. written in the upper camel case style).
Metadata Display Name	It is RECOMMENDED to use the ·Metadata Identifier· with spaces between the words.
Metadata Description	The ·Description· of the ·SiLA Client Metadata· SHOULD give an accurate, in depth depiction of what the ·SiLA Client Metadata· is.

Examples

[COMPLETE; as of 0.2]

The following two examples of ·Feature· designs illustrate the use of ·SiLA Client Metadata· in different categories:

A “Lock Controller” Feature

[COMPLETE; as of 0.2]

The Feature design specifies the following:

- A ·Command· “LockServer” to lock a ·SiLA Server· with a ·Command Parameter· “LockIdentifier” in the form of a ·SiLA String Type·
- A ·SiLA Client Metadata· “LockIdentifier” in the form of a ·SiLA String Type·, to be sent with ·Commands· while the ·SiLA Server· is locked.

The ·SiLA Client· locks a ·SiLA Server· via ·Command· “LockServer” providing a lock ·Identifier·. Any future call to any (lock protected) ·Command· (or all ·Commands· of any lock protected ·Feature·) will require the ·SiLA Client· to send a ·SiLA Client Metadata· with the ·Identifier·

"LockIdentifier" and a value of the lock ·Identifier· provided with the "LockServer" ·Command· - otherwise accessing the respective ·Command· is not possible.

Authorization Service

[COMPLETE; as of 0.2]

The ·Feature· design specifies the following:

- A ·Command· "Authorize" to get an authorization token in the form of a ·SiLA String Type·.
- A ·SiLA Client Metadata· "AuthorizationToken" in the form of a ·SiLA String Type·.

The ·SiLA Client· obtains an authorization token via ·Command· "Authorize"; once successfully obtained, any future call to any ·Command· or ·Property· (except for the ·SiLA Service Feature·) will require the ·SiLA Client· to send a ·SiLA Client Metadata· with the ·Identifier· "AuthorizationToken" and a value of the authorization token obtained - otherwise accessing the ·SiLA Server· is not possible. All ·Commands· and all ·Properties· would be affected in this design.

A "Gateway" Feature

[COMPLETE; as of 1.1]

The ·Feature· design specifies the following:

- A ·Command· to get the ·SiLA Server UUID·s of the connected ·SiLA Servers·.
- A ·SiLA Client Metadata· containing the ·SiLA Server UUID· of the selected ·SiLA Server·.

The ·SiLA Client· obtains a list of all connected ·SiLA Servers· from the gateway. The gateway itself is a ·SiLA Server· that implements and "relays" all the ·Features· of all connected ·SiLA Servers· to the ·SiLA Client· connected to the gateway. In order for the ·SiLA Client· to be able to select the "correct" ·SiLA Server· behind the gateway, the ·SiLA Client· makes use of the ·SiLA Client Metadata· to select the right server (through the ·SiLA Server UUID·).

Defined Execution Errors in Metadata Context

[*COMPLETE; as of 0.2*]

One or more ·Defined Execution Errors· MAY be specified for a ·SiLA Client Metadata·. The ·Defined Execution Errors· are defined in the context of the ·Feature· that defines the ·SiLA Client Metadata· and can be re-used within the ·Feature·, e.g. among different ·SiLA Client Metadata·.

Item	Description
Defined Execution Errors	A list of ·Defined Execution Error Identifiers· referring to all ·Defined Execution Errors· that can occur when calling a ·Command· or accessing (reading or subscribing to) a ·Property· which is affected by a certain ·SiLA Client Metadata·.

Defined Execution Errors

[COMPLETE; as of 0.2]

A ·Feature Designer· MAY specify ·Defined Execution Errors·. Using ·Defined Execution Errors·, a ·Feature Designer· is able to design specific error conditions. A ·SiLA Client· in turn MAY use this information to handle ·Defined Execution Errors· in a more sophisticated way as for instance

·Undefined Execution Errors· with the knowledge about the nature of the error from the ·Feature· design.

·Defined Execution Errors· MAY be assigned to one or more ·Commands·, ·Properties· or ·SiLA Client Metadata·.

For more details, please refer to chapter Error Categories.

Item	Description
Defined Execution Error Identifier	[Definition: Defined Execution Error Identifier] The **Defined Execution Error Identifier** is the ·Identifier· of a ·Defined Execution Error·. A **Defined Execution Error Identifier** MUST be unique within the scope of a ·Feature·. Uniqueness MUST be checked without taking lower and upper case into account, see Uniqueness of Identifiers.
Defined Execution Error Display Name	[Definition: Defined Execution Error Display Name] The **Defined Execution Error Display Name** is the ·Display Name· of a ·Defined Execution Error·.
Defined Execution Error Description	[Definition: Defined Execution Error Description] The **Defined Execution Error Description** is the ·Description· of a ·Defined Execution Error·.

Best Practice

[COMPLETE; as of 0.2]

A ·Feature Designer· SHOULD adhere to the following guidelines:

Item	Guidelines
Defined Execution Error Identifier	The ·Identifier· of a ·Defined Execution Error· SHOULD be in CamelCase (i.e. written in the upper camel case style). The ·Defined Execution Error Identifier· SHOULD be in the form: **AdjectiveNoun(s)** (adjective, singular or plural noun(s), depending on context).
Defined Execution Error Display Name	A human readable name for the ·Defined Execution Error·. It is RECOMMENDED to use the ·Defined Execution Error Identifier· with spaces between separate words.
Defined Execution Error Description	The ·Defined Execution Error Description· SHOULD provide an accurate description of the ·Defined Execution Error·.

Example

[COMPLETE; as of 0.2]

For examples regarding ·Defined Execution Errors·, please refer to chapter Error Categories.

Custom Data Types

[COMPLETE; as of 0.1]

[Definition: Custom Data Type] A **Custom Data Type** allows to assign a custom ·Fully Qualified Custom Data Type Identifier·, ·Custom Data Type Display Name· and ·Custom Data Type Description· to a ·SiLA Data Type·. ·SiLA Data Types· that have been defined in this way can be used as ·SiLA Data Types· of ·Parameters·, ·Command Responses·, ·Intermediate Command Responses·, ·Properties· or ·SiLA Client Metadata· like any other ·SiLA Data Type·.

Item	Description
Custom Data Type Identifier	[Definition: Custom Data Type Identifier] A **Custom Data Type Identifier** is the ·Identifier· of a ·SiLA Data Type·. A **Custom Data Type Identifier** MUST be unique within the scope of a ·Feature·. The ·Identifiers· of the ·SiLA Basic Types· are reserved and MUST NOT be used.
Custom Data Type Display Name	[Definition: Custom Data Type Display Name] A **Custom Data Type Display Name** is the ·Display Name· of a ·SiLA Data Type·.
Custom Data Type Description	[Definition: Custom Data Type Description] A **Custom Data Type Description** Is the ·Description· of a ·SiLA Data Type·.
Custom Data Type Data Type	[Definition: Custom Data Type Data Type] A **Custom Data Type Data Type** is the ·SiLA Data Type· of a ·Custom Data Type·.

Best Practice

[COMPLETE; as of 0.1]

A ·Feature Designer· SHOULD adhere to the following guidelines:

Item	Guidelines
Custom Data Type Identifier	The ·Identifier· of a ·Custom Data Type· SHOULD be in CamelCase (i.e. written in the upper camel case style).
Custom Data Type Display Name	A human readable name for the ·SiLA Data Type·. It is RECOMMENDED to use the SiLA ·Fully Qualified Custom Data Type Identifier· with spaces between separate words.
Custom Data Type Description	The ·Custom Data Type Description· SHOULD provide an accurate description of the ·SiLA Data Type·.

SiLA Data Types

[COMPLETE; as of 0.1]

This section describes the ·SiLA Data Type· system.

[Definition: SiLA Data Type] A **SiLA Data Type** describes the data type of any information exchanged between ·SiLA Client· and ·SiLA Server·. A **SiLA Data Type** MUST either be a ·SiLA Basic Type· or a ·SiLA Derived Type·. A **SiLA Data Type** is used to describe the content communicated in:

- a ·Parameter· of a ·Command·
- a ·Command Response· or ·Intermediate Command Responses·

- a ·Property· value
- a ·SiLA Client Metadata·

SiLA Basic Types

[COMPLETE; as of 1.0]

[Definition: SiLA Basic Type] ·SiLA Data Types· are separated into **SiLA Basic Types** and ·SiLA Derived Types·. The following **SiLA Basic Types** are predefined by SiLA.

[Definition: SiLA Numeric Type] The ·SiLA Integer Type· and the ·SiLA Real Type· are **SiLA Numeric Types**.

Identifier	Description
String	[Definition: SiLA String Type] The **SiLA String Type** represents a plain text composed of maximum 2 x 2^{20} UNICODE characters. Use the ·SiLA Binary Type· for larger data. It is RECOMMENDED to specify a ·Constraint·, e.g. a ·Content Type Constraint· or ·Schema Constraint· for the ·SiLA String Type· in order to make the string content type safe.
Integer	[Definition: SiLA Integer Type] The **SiLA Integer Type** represents an integer number within a range from the minimum value of -2^{63} up to the maximum value of $2^{63}-1$. This is a ·SiLA Numeric Type·.
Real	[Definition: SiLA Real Type] The **SiLA Real Type** represents a real number as defined per IEEE 754 double-precision floating-point number. This is a ·SiLA Numeric Type·.
Boolean	[Definition: SiLA Boolean Type] The **SiLA Boolean Type** represents a Boolean value. This is a ·SiLA Data Type· representing one of two possible values, usually denoted as true and false.
Binary	[Definition: SiLA Binary Type] The **SiLA Binary Type** represents arbitrary binary data of any size such as images, office files, etc. If the ·SiLA Binary Type· is used for character data, e.g. plain text, XML or JSON, the character encoding MUST be UTF-8. It is RECOMMENDED to specify ·Constraint·, e.g. a ·Content Type Constraint· or ·Schema Constraint· for the ·SiLA Binary Type· in order to make the binary content type safe.
Date[1)]	[Definition: SiLA Date Type] The **SiLA Date Type** represents a ISO 8601 date (year [1–9999][3)], month [1–12], day [1–31]) in the Gregorian calendar, with an additional timezone (as an offset from UTC). A **SiLA Date Type** consists of the top-open interval of exactly one day in length, beginning on the beginning moment of each day (in each timezone), i.e. '00:00:00', up to but not including '24:00:00' (which is identical with '00:00:00' of the next day).

Time[1),2)]	[Definition: SiLA Time Type] The **SiLA Time Type** represents a ISO 8601 time (hours [0–23], minutes [0–59], seconds [0–59], milliseconds [0–999], with an additional timezone [as an offset from UTC]).
Timestamp[1),2)]	[Definition: SiLA Timestamp Type] The **SiLA Timestamp Type** represents both, ISO 8601 date and time in one (year [1–9999][3)], month, day, hours [0–23], minutes [0–59], seconds [0–59], milliseconds [0–999], with an additional timezone [as an offset from UTC]).
Any	[Definition: SiLA Any Type] The **SiLA Any Type** represents information that can be of any ·SiLA Data Type·, except for a ·Custom Data Type· (i.e. the **SiLA Any Type** MUST NOT represent information of a ·Custom Data Type·). The value of a **SiLA Any Type** MUST contain both the information itself and the ·SiLA Data Type·. Note: The ·SiLA Any Type· may be used to be able to communicate data whose ·SiLA Data Type· is unknown at design time of the ·Feature·. This ·SiLA Data Type· should be used with care, as it not only adds overhead to the communication, but it also adds complexity for the implementation of both, ·SiLA Servers· and ·SiLA Clients·.
Void	[Definition: SiLA Void Type] The **SiLA Void Type** represents no data. It MUST only be used as a value of the ·SiLA Any Type·. Example: The ·SiLA Void Type· allows the implementer of a ·SiLA Server· that offers a ·Feature· containing a ·Command· that returns a ·SiLA Any Type· to return “no” ·Command Response·.

Additional Details on Time, Date and Timestamp

[COMPLETE; as of 0.1]

[1)] SiLA 2 defines three ·SiLA Data Types· dealing with a point in time:

- The ·SiLA Timestamp Type· defines an absolute point in time.
- The ·SiLA Date Type· defines an absolute day in time.
- The ·SiLA Time Type· is an absolute time within an arbitrary day. The ·SiLA Date Type· and ·SiLA Time Type· can be useful e.g. in a scheduling context (“Run five times on Dec 5, 2017”, “Run at 04:17”).

[2)] A ·Feature Designer· MAY define a new ·SiLA Data Type· with higher time resolution within a ·Feature Definition· if needed. For the time being, the SiLA working group does not see a need for such a data type being defined in SiLA 2 as a ·SiLA Basic Type· or pre-defined ·SiLA Derived Type·.

[3)] The year must be in the range from year 1 to year 9999. Representing years before 1 is not possible with SiLA, as no use case could be found for this.

Time Durations

[COMPLETE; as of 0.1]

A note on time duration: It is recommended to use a ·SiLA Numeric Type· with a ·Unit Constraint· to design a time duration.

Best Practice

[COMPLETE; as of 0.1]

It is RECOMMENDED that a ·Feature Designer· specifies a ·Unit Constraint· for every ·SiLA Numeric Type· that represents a physical quantity.

It is RECOMMENDED that a ·Feature Designer· specifies a ·Content Type Constraint· or a ·Schema Constraint· for every ·SiLA Binary Type· used.

SiLA Derived Types

[COMPLETE; as of 0.1]

[Definition: SiLA Derived Type] There are three different **SiLA Derived Types** defined:

- ·SiLA List Type· is a list with entries of the same ·SiLA Data Type·.
- ·SiLA Structure Type· is a structure composed of named elements with the same or different ·SiLA Data Types·.
- ·SiLA Constrained Type· is a ·SiLA Basic Type· or a ·SiLA List Type· with ·Constraints·.

SiLA List Type

[COMPLETE; as of 0.1]

[Definition: SiLA List Type] The **SiLA List Type** is an ordered list with entries of the same ·SiLA Data Type·.

The ·SiLA Data Type· of the list entries MUST be specified as follows:

Item	Description
SiLA Data Type	A ·SiLA Data Type·.

The ·SiLA Data Type· of the list entries MUST NOT be a ·SiLA List Type· itself (list of list or nesting of lists is not allowed).

SiLA Structure Type

[COMPLETE; as of 0.1]

[Definition: SiLA Structure Type] The **SiLA Structure Type** is a structure composed of one or more named elements with the same or different ·SiLA Data Types·.

The following information MUST be provided for each element of the structure:

Item (for each element)	**Description**
Element Identifier	[Definition: Element Identifier] The **Element Identifier** is the ·Identifier· of this element of the structure. The ·Identifier· MUST be

	unique within the scope of a given ·SiLA Structure Type·. Uniqueness MUST be checked without taking lower and upper case into account, see Uniqueness of Identifiers.
Element Display Name	[Definition: Element Display Name] The **Element Display Name** is the ·Display Name· of an element of the structure.
Element Description	[Definition: Element Description] The **Element Description** is the ·Description· of an element of the structure.
Element Data Type	[Definition: Element Data Type] The **Element Data Type** is the ·SiLA Data Type· of an element of the structure.

Best Practice

[COMPLETE; as of 0.1]

A ·Feature Designer· SHOULD adhere to the following guidelines:

Item	Guidelines
Element Identifier	The ·Element Identifier· SHOULD be in CamelCase (i.e. written in the upper camel case style).
Element Display Name	It is RECOMMENDED to use the ·Identifier· with spaces between separate words.
Element Description	The ·Element Description· SHOULD provide an accurate description of this element of the structure.

SiLA Structure Type vs. SiLA Binary Type

[COMPLETE; as of 0.1]

SiLA 2 is strongly typed, it is therefore RECOMMENDED to use ·SiLA Data Types· where it makes sense. The ·SiLA Binary Type· SHOULD be used for data for which a proper, open specification exists (such as image formats, …) and for which it would be difficult to handle it as a ·SiLA Data Type·.

Structure Type vs. Using AnIML

[COMPLETE; as of 0.1]

When dealing with data from analytical instruments, SiLA recommends to use AnIML instead of the ·SiLA Structure Type·.

SiLA Constrained Type

[COMPLETE; as of 0.1, updates: 1.1]

[Definition: SiLA Constrained Type] The **SiLA Constrained Type** is a ·SiLA Data Type· with one or more ·Constraints· that act as a logical AND. The **SiLA Constrained Type** MUST be based on either a ·SiLA Basic Type· or a ·SiLA List Type·~~, or a ·SiLA Constrained Type~~·. The ·Constraints· in the type itself and the type it is based on are to act together as a logical conjunction (AND).

[Definition: Constraint] A **Constraint** limits the allowed value, size, range, etc. that a ·SiLA Data Type· can assume. A ·SiLA Server· MUST check the all the **Constraints** and issue a ·Validation Error· if ·Constraints· are violated.

The ·SiLA Data Type· that will be constrained MUST be specified as follows:

Item	Description
SiLA Data Type	A ·SiLA Basic Type· or a ·SiLA List Type·.

Then, the following information MUST be provided for a ·Constraint· for a given ·SiLA Constrained Type·:

Item	Description
Constraint Identifier	[Definition: Constraint Identifier] A **Constraint Identifier** is the ·Identifier· of the ·Constraint·. The **Constraint Identifier** MUST be one of the identifiers defined in the two tables below: Constraints to SiLA Basic Types and Constraints to SiLA List Type.
Constraint Value	[Definition: Constraint Value] The **Constraint Value** is the actual parameterization of the ·Constraint·.

Constraints to SiLA Basic Types

[COMPLETE; as of 1.0]

The following ·Constraints· are possible for a ·SiLA Basic Type·.

Note that no ·Constraints· are applicable to the ·SiLA Boolean Type·.

Constraint Identifier	Description	Applicable to SiLA Basic Type(s)
Length	[Definition: Length Constraint] A **Length Constraint** specifies the exact number of characters allowed. The ·Constraint Value· MUST be an integer number equal or greater than zero (0) up to the maximum value of 2^{63}-1.	·SiLA String Type·, ·SiLA Binary Type·
Minimal Length	[Definition: MinimalLength Constraint] A **Minimal Length Constraint** specifies the minimum number of characters (for a ·SiLA String Type·) or bytes (for a ·SiLA Binary Type·) allowed. The ·Constraint Value· MUST be an integer number equal or greater than zero (0) up to the maximum value of 2^{63}-1.	·SiLA String Type·, ·SiLA Binary Type·
Maximal Length	[Definition: Maximal Length Constraint] A **Maximal Length Constraint** specifies the maximum number of characters (for a ·SiLA String Type·) or bytes (for a ·SiLA Binary Type·) allowed. The ·Constraint Value· MUST be an	·SiLA String Type·, ·SiLA Binary Type·

	integer number greater than zero (0) up to the maximum value of 2^{63}-1.	
Set	[Definition: Set Constraint] A **Set Constraint** defines a set of acceptable values for a given ·SiLA Basic Type·. The list of acceptable ·Constraint Values· must have the same ·SiLA Data Type· as the ·SiLA Basic Type· that this ·Constraint· applies to.	·SiLA String Type·, ·SiLA Numeric Type·, ·SiLA Date Type·, ·SiLA Time Type·, ·SiLA Timestamp Type·[1)]
Pattern	[Definition: Pattern Constraint] A **Pattern Constraint** defines the exact sequence of characters that are acceptable, as specified by a so-called regular expression. The ·Constraint Value· MUST be a XML Schema Regular Expression (Regular Expressions Quick Start).	·SiLA String Type·
Maximal Exclusive	[Definition: Maximal Exclusive Constraint] A **Maximal Exclusive Constraint** specifies the upper bounds for ·SiLA Numeric Types· (the value which is constrained MUST be less than this ·Constraint·) and ·SiLA Date Type·, ·SiLA Time Type· and ·SiLA Timestamp Type· (the value which is constrained MUST be before this ·Constraint·). The ·Constraint Value· must be of the same ·SiLA Data Type· as the ·SiLA Basic Type· that this ·Constraint· applies to.	·SiLA Numeric Type·, ·SiLA Date Type·, ·SiLA Time Type·, ·SiLA Timestamp Type·[1)]
Maximal Inclusive	[Definition: Maximal Inclusive Constraint] A **Maximal Inclusive Constraint** specifies the upper bounds for ·SiLA Numeric Types· (the value which is constrained MUST be less than or equal to this ·Constraint·) and ·SiLA Date Type·, ·SiLA Time Type· and ·SiLA Timestamp Type· (the value which is constrained MUST be before or at this ·Constraint·). The ·Constraint Value· must be of the same ·SiLA Data Type· as the ·SiLA Basic Type· that this ·Constraint· applies to.	·SiLA Numeric Type·, ·SiLA Date Type·, ·SiLA Time Type·, ·SiLA Timestamp Type·[1)]
Minimal Exclusive	[Definition: Minimal Exclusive Constraint] A **Minimal Exclusive Constraint** specifies the lower bounds for ·SiLA Numeric" Types· (the value which is constrained MUST be greater than this ·Constraint·) and ·SiLA Date Type·, ·SiLA Time Type· and ·SiLA Timestamp Type· (the value which is constrained MUST be after this ·Constraint·). The ·Constraint Value· must be of the same ·SiLA Data Type· as the ·SiLA Basic Type· that this ·Constraint· applies to.	·SiLA Numeric Type·, ·SiLA Date Type·, ·SiLA Time Type·, ·SiLA Timestamp Type·[1)]
Minimal Inclusive	[Definition: Minimal Inclusive Constraint] A **Minimal Inclusive Constraint** specifies the lower bounds for ·SiLA Numeric Types· (the value	·SiLA Numeric Type·, ·SiLA Date Type·, ·SiLA Time

	which is constrained MUST be greater than or equal to this ·Constraint·) and ·SiLA Date Type·, ·SiLA Time Type· and ·SiLA Timestamp Type· (the value which is constrained MUST be at or after this ·Constraint·). The ·Constraint Value· must be of the same ·SiLA Data Type· as the ·SiLA Basic Type· that this ·Constraint· applies to.	Type·, ·SiLA Timestamp Type·[1)]
Unit	[Definition: Unit Constraint] A **Unit Constraint** specifies the unit of a physical quantity, see Unit Constraint for a definition of the allowed ·Constraint Values·.	·SiLA Integer Type·, ·SiLA Real Type·
Content Type	[Definition: Content Type Constraint] A **Content Type Constraint** specifies the type of content of a binary or textual ·SiLA Data Type· based on a RFC 2045 ContentType, see Content Type Constraint for a definition of the allowed ·Constraint Values·.	·SiLA Binary Type·, ·SiLA String Type·
Fully Qualified Identifier	[Definition: Fully Qualified Identifier Constraint] A **Fully Qualified Identifier Constraint** specifies the content of the ·SiLA String Type· to be a ·Fully Qualified Identifier· and indicates the type of the identifier. Note that this is comparable to a ·Pattern Constraint·; that is, the content is not required to actually identify something, it just has to be a semantically correct ·Fully Qualified Identifier·. The ·Constraint Value· MUST be exactly one of this list: • “FeatureIdentifier”, • “CommandIdentifier”, • “CommandParameterIdentifier”, • “CommandResponseIdentifier”, • “IntermediateCommandResponseIdentifier”, • “DefinedExecutionErrorIdentifier”, • “PropertyIdentifier”, • “DataTypeIdentifier” • “MetadataIdentifier”	·SiLA String Type·
Schema	[Definition: Schema Constraint] A **Schema Constraint** specifies the type of content of a binary or textual ·SiLA Data Type· based on a schema, see Schema Constraint for a definition of the allowed ·Constraint Values·..	·SiLA Binary Type·, ·SiLA String Type·
Allowed Types	[Definition: Allowed Types Constraint] An **Allowed Types Constraint** defines a list of ·SiLA Data Types· that the ·SiLA Any Type· is allowed to represent. The ·Constraint Value· MUST be a list of ·SiLA Data Types·, but MUST NOT be a ·Custom Data Type· or a ·SiLA Derived Type· containing a ·Custom Data Type·.	·SiLA Any Type·

Additional Details on Representing Constraint's Data in the Feature Definition Language

[COMPLETE; as of 1.1]

[1)] Use the following XSD schema types to serialize the ·Constraint Value· of the following ·SiLA Constrained Types· into an XML representation of the Feature Definition Language:

- ·SiLA String Type· → `xs:string`
- ·SiLA Numeric Type· → `xs:double`
- ·SiLA Date Type· → `xs:date` with explicit timezone fragment
- ·SiLA Time Type· → `xs:time` with explicit timezone fragment
- ·SiLA Timestamp Type· → `xs:dateTime` with explicit timezone fragment

Also refer to the annotations in Constraints.xsd.

Content Type Constraint

[COMPLETE; as of 0.2]

The ·Content Type Constraint· SHOULD be used to identify the exact nature of a textual or binary value. The ·Constraint Value· MUST contain the "type" and "subtype" as specified in chapter "Syntax of the Content-Type Header Field" from RFC 2045. The "type" and "subtype" elements are always matched case-insensitive. Specific rules apply regarding valid values for the elements "type", "subtype" and "parameter", as stated below.

If the ·Content Type Constraint· constrains a ·SiLA Binary Type· that contains textual data (e.g. XML or JSON), the character encoding of this data MUST be UTF-8.

It is RECOMMENDED to only refer to registered media type and subtype names as listed by IANA under IANA Media Types.

The Content Types officially supported by SiLA can be found on SiLA Supported Content Types and MUST be handled properly by all implementations of ·SiLA Servers·.

Schema Constraint

[COMPLETE; as of 0.2]

The ·Schema Constraint· SHOULD be used to identify the exact nature of a textual or binary value. The ·Schema Constraint· specifies the schema with which the data MUST be compliant with.

The ·Constraint Value· is made up of the three following components, of which the "Type" and one of the components "Inline" or "URL" MUST always be provided:

- Type: schema type value can be "Xml" (W3C XML Schema) or "Json" (Json Schema)
- Inline: Schema content is provided inline, as the value of the "Inline" component
- URL (according to RFC 3986): Schema content is provided as a URL (as the value of the "URL" component). It is RECOMMENDED to provide a URL pointing to a document served with an "Access-Control-Allow-Origin" header that permits cross-origin use.

If the ·Schema Constraint· constrains XML or JSON data in a ·SiLA Binary Type·, the character encoding of this data MUST be UTF-8.

Note: In case the schema is provided inline as part of a ·Feature Definition·, proper escape sequences or mechanisms, as specified for XML, have to be used.

Unit Constraint

[COMPLETE; as of 1.0]

The ·Unit Constraint· specifies the physical unit for a specific ·SiLA Integer Type· or ·SiLA Real Type· with

- a human readable label,
- conversion information to convert the value into SI units and
- the actual SI unit.

The label allows a ·SiLA Client· to present values in a nice and human readable way. The conversion factor and offset allow a ·SiLA Client· or ·SiLA Server· to convert a value from the unit indicated by the label into SI units or vice versa. The relation to SI units allows a ·SiLA Client· or ·SiLA Server· to programmatically interpret the actual SI unit of the value in order to perform proper calculations with physical values.

If a ·Unit Constraint· is applied to a ·SiLA Data Type·, a ·SiLA Client· or ·SiLA Server· MUST always send the value of that ·SiLA Data Type· in the unit corresponding to the ·Unit Label·.

The ·Constraint Value· is made up of the following items:

Item	Description
Unit Label	[Definition: Unit Label] The **Unit Label** is the arbitrary label denoting the physical unit that the ·Unit Constraint· defines. The **Unit Label** MUST be a string of UNICODE characters up to a maximum of 255 characters in length.
Conversion Factor	[Definition: Conversion Factor] The **Conversion Factor** specifies the conversion from the unit with a given ·Unit Label· into SI units, according to the definition in chapter Unit Conversion. The **Conversion Factor** MUST be a real number as defined per IEEE 754 double-precision floating-point number.
Conversion Offset	[Definition: Conversion Offset] The **Conversion Offset** specifies the conversion from the unit with a given ·Unit Label· into SI units, according to the definition in chapter Unit Conversion. The **Conversion Offset** MUST be a real number as defined per IEEE 754 double-precision floating-point number.
[Definition: SI Base Unit] The **SI Base Unit** is the combination of SI units that specifies the same base quantity or dimension as indicated by the ·Unit Label·. The following items MUST be provided for each SI unit N, that makes up the **SI Base Unit** (N is an integer number in the range of 1 – 8).	
SI Unit Name N	[Definition: SI Unit Name] The **SI Unit Name** is a name referring to an SI unit and MUST be either: ● “Dimensionless”, ● “Meter”, ● “Kilogram”, ● “Second”, ● “Ampere”, ● “Kelvin”, ● “Mole”

	or • “Candela”
SI Unit Exponent N	[Definition: SI Unit Exponent] The **SI Unit Exponent** is the exponent to apply to the SI unit corresponding to ·SI Unit Name· and must be an integer number within a range from the minimum value of -2^{63} up to the maximum value of $2^{63}-1$.

Unit Conversion

[COMPLETE; as of 1.0]

Let x_{orig} be the value of a ·SiLA Constrained Type· with a ·Unit Constraint·. According to the ·Unit Constraint·, the value x_{orig} has to be in a unit as specified by the corresponding ·Unit Label·.

Let x_{conv} be the same value as x_{orig}, but being in the corresponding ·SI Base Unit· (i.e. being of the same base quantity or dimension as the ·Unit Label· of x_{orig}).

The relation between x_{orig} and x_{conv} is:

$$x_{conv} = x_{orig} \cdot factor + offset$$

where

$x_{conv}[SI\ Base\ Unit]$ is the value converted to the ·SI Base Unit·

$x_{orig}[Unit\ Label]$ is the original value in the unit as defined by the ·Unit Label·

The ·SI Base Unit· is made up of ·SI Unit Names· and ·SI Unit Exponents· as follows:

$$[SI\ Base\ Unit] = [\prod_{N} SI\ Unit\ Name_{N}^{SI\ Unit\ Exponent_{N}}]$$

Best Practice

[COMPLETE; as of 1.0]

It is RECOMMENDED that a ·Feature Designer· specifies a ·Unit Constraint· for each ·SiLA Integer Type· or ·SiLA Real Type· that represents a physical quantity.

Item	Description
·Unit Label·	It is RECOMMENDED, as a guideline, to define the ·Unit Label· according to the “Name” or “Symbol” column, as specified in the tables of the section “Derived units with special names” in the Wikipedia article about SI derived units. The ·Unit Label· is considered a human readable text and MUST therefore adhere to the SiLA conventions for Internationalization.
·Conversion Factor· ·Conversion Offset·	As a guidance, conversion factors from this Wikipedia article about conversion of units can be taken as a basis.

Examples

[COMPLETE; as of 1.0]

Consider the following simple example for nanometer:

·Unit Label· = "nm"

·Conversion Factor· = 10^{-9}

·Conversion Offset· = 0

·SI Unit Name·$_1$ = “Meter”

·SI Unit Exponent·$_1$ = 1

Hence, the above formula for the conversion reads:

$$x_{conv} = x_{orig} \cdot 10^{-9} + 0$$

x_{conv} being in ·SI Base Unit· $[Meter^{1}]$

x_{orig} in $[nm]$.

Another example for a volume in cubic centimetres (ccm):

·Unit Label· = "ccm"

·Conversion Factor· = 10^{-6}

·Conversion Offset· = 0

·SI Unit Name·$_1$ = “Meter”

·SI Unit Exponent·$_1$ = 3

Hence, the above formula for the conversion reads:

$$x_{conv} = x_{orig} \cdot 10^{-6} + 0$$

x_{conv} being in ·SI Base Unit· $[Meter^{3}]$

x_{orig} in $[ccm]$

A force in Newton (N):

·Unit Label· = "N"

·Conversion Factor· = 1

·Conversion Offset· = 0

·SI Unit Name·$_1$ = “Kilogram”

·SI Unit Exponent·$_1$ = 1

·SI Unit Name·$_2$ = “Meter”

·SI Unit Exponent·$_2$ = 1

·SI Unit Name·$_3$ = “Second”

·SI Unit Exponent·$_3$ = -2

Hence, the above formula for the conversion reads:

$x_{conv} = x_{orig} \cdot 1 + 0$

x_{conv} being in ·SI Base Unit· $[Kilogramm^{1} Meter^{1} Second^{-2}]$

x_{orig} in $[N]$

Likewise for degrees Celsius or degrees Fahrenheit:

·Unit Label· = "°C"

·Conversion Factor· = 1

·Conversion Offset· = 273.15

·SI Unit Name·$_1$ = “Kelvin”

·SI Unit Exponent·$_1$ = 1

Hence, the above formula for the conversion reads:

$x_{conv} = x_{orig} \cdot 1 + 273.15$

x_{conv} being in ·SI Base Unit· $[Kelvin^{1}]$

x_{orig} in $[°C]$.

Or for °F:

·Unit Label· = "°F"

·Conversion Factor· = $\frac{5}{9}$

·Conversion Offset· = 255.37

·SI Unit Name·$_1$ = “Kelvin”

·SI Unit Exponent·$_1$ = 1

Hence, the above formula for the conversion reads:

$x_{conv} = x_{orig} \cdot \frac{5}{9} + 255.37$

x_{conv} being in ·SI Base Unit· $[Kelvin^{1}]$

x_{orig} in $[°F]$.

Rotational speed in rpm:

·Unit Label· = "rpm"

·Conversion Factor· = $\frac{1}{60}$

·Conversion Offset· = 0

·SI Unit Name·$_1$ = “Second”

·SI Unit Exponent·$_1$ = -1

Hence, the above formula for the conversion reads:

$x_{conv} = x_{orig} \cdot \frac{1}{60} + 0$

x_{conv} being in ·SI Base Unit· $[Second^{-1}]$

x_{orig} in $[rpm]$.

Constraints to SiLA List Type

[COMPLETE; as of 0.1]

The following ·Constraints· are possible for the ·SiLA List Type·:

Identifier	Description
Element Count	[Definition: Element Count Constraint] An **Element Count Constraint** specifies the exact number of elements that a list MUST have. The ·Constraint Value· MUST be an integer number equal or greater than zero (0) up to the maximum value of 2^{63}-1.
Minimal Element Count	[Definition: Minimal Element Count Constraint] An **Minimal Element Count Constraint** specifies the exact number of elements that a list MUST have in minimum. The ·Constraint Value· MUST be an integer number equal or greater than zero (0) up to the maximum value of 2^{63}-1.
Maximal Element Count	[Definition: Maximal Element Count Constraint] An **Maximal Element Count Constraint** specifies the exact number of elements that a list MUST have in maximum. The ·Constraint Value· MUST be an integer number equal or greater than zero (0) up to the maximum value of 2^{63}-1.

Feature Attributes

[COMPLETE; as of 0.1]

[Definition: Attribute] **Attributes** are a part of a ·Feature Definition·. **Attributes** are not necessary for the implementation of the ·Features· themselves, but help to maintain them. The following **Attributes** exist and are mandatory: ·SiLA 2 Version·, ·Feature Version·, ·Maturity Level·, ·Originator· and ·Category·.

SiLA 2 Version

[COMPLETE; as of 0.2, update: 1.1]

[Definition: SiLA 2 Version] The version of the SiLA 2 Specification that this ·Feature· was developed against. Any ·SiLA Server· or ·SiLA Client· that was developed against a **SiLA 2 Version** with a ·Major SiLA 2 Version· of "1" or larger MUST be able to interoperate with each other (backwards and forwards compatibility). The **SiLA 2 Version** is a combination of the ·Major SiLA 2 Version· and the ·Minor SiLA 2 Version·, separated by a dot (.).

Item	Description
Major SiLA 2 Version	[Definition: Major SiLA 2 Version] The **Major SiLA 2 Version**. MUST be an integer greater or equal than zero (0).

Minor SiLA 2 Version	[Definition: Minor SiLA 2 Version] The **Minor SiLA 2 Version**. MUST be an integer greater or equal than zero (0).

As of this release of the SiLA 2 Standard, the ·SiLA 2 Version· SHALL be “1.1” (i.e. the ·Major SiLA 2 Version· SHALL be 1 and the ·Minor SiLA 2 Version· SHALL be 1).

Feature Version

[COMPLETE; as of 0.1]

[Definition: Feature Version] Any ·Feature· MUST specify a **Feature Version** so that different versions of a ·Feature· can be distinguished during its life cycle. A **Feature Version** consists of a ·Major Feature Version· and a ·Minor Feature Version·. A ·Feature· MUST specify both a ·Major Feature Version· and a ·Minor Feature Version·.

A ·Feature· SHOULD be backwards and forwards compatible within the same ·Major Feature Version·. This means that a ·SiLA Client·, implemented by referencing a ·Feature· with a newer ·Minor Feature Version·, is RECOMMENDED be able to also use a ·SiLA Server· that offers this ·Feature· with an older ·Minor Feature Version· and the same ·Major Feature Version· without issues. This SHOULD also work vice versa for a ·SiLA Client· implemented against an older ·Minor Feature Version· accessing a ·SiLA Server· offering that ·Feature· in a newer ·Minor Feature Version·.

Item	Description
Major Feature Version	[Definition: Major Feature Version] The **Major Feature Version** of a ·Feature·. MUST be an integer greater or equal than zero (0).
Minor Feature Version	[Definition: Minor Feature Version] The **Minor Feature Version** of a ·Feature·. MUST be an integer greater or equal than zero (0).

SiLA’s versioning requirements go beyond versioning of ·Features·, please also see Versioning Strategy on more details about versioning the specification documents.

The following changes to a ·Feature Definition· MUST increase the ·Major Feature Version·:

- Renaming any ·Identifier·
- Changing any ·Display Name·, if the original behavior changes.
- Modifying any ·Description· in a way that the original behavior changes.
- Removing ·Commands· or ·Properties·.
- Removing ·Parameters·, ·Intermediate Command Responses· or ·Command Responses·.
- Changing the ·SiLA Data Type· of any item in a ·Feature·.

The following changes to a ·Feature Definition· MAY increase the ·Major Feature Version·:

- Changing any ·Display Name·, if the original behavior does not change.
- Modifying any ·Description· in a way that the original behavior does not change.
- Adding new ·Commands· or ·Properties·.
- Adding ·Parameters·, ·Intermediate Command Responses· or ·Command Responses·

Changes in any ·Description· or ·Display Name· that do not modify the behavior that the ·Feature· describes but only have an impact on their documentation, such as clarifying the ·Feature

Description· or correcting spelling mistakes MUST increase the ·Minor Feature Version· but usually SHALL not increase the ·Major Feature Version·.

Maturity Level

[COMPLETE; as of 0.1]

Item	Description
Maturity Level	Enum, see table below. This means that it can only be one of the chosen words of the table below.

[Definition: Maturity Level] SiLA 2 defines the following **Maturity Levels**, in order of increasing maturity:

Maturity Level	Description
Draft	[Definition: Draft] The ·Maturity Level· **Draft** means that the ·Feature· is in a development state.
Verified	[Definition: Verified] The ·Maturity Level· **Verified** means that the ·Feature· has been verified as meeting the normative part of the SiLA 2 specification, including the best practices.
Normative	[Definition: Normative] The ·Maturity Level· **Normative** means that the ·Feature· is now considered stable and has been subject to a round of formal balloting. This ·Maturity Level· MUST only be applied to ·Features· with ·Originator· “org.silastandard”. The requirements of a ·Verified· ·Feature· also apply to **Normative** ·Features·.

Originator

[COMPLETE; as of 0.1]

[Definition: Originator] The **Originator** is a text identifying the organization who created and owns a ·Feature·.

Item	Description
Originator	The ·Originator· MUST consist of one word or words separated by dots (“.”). Each word MUST start with a lower-case letter (a-z), followed by any number of lower-case letters (a-z) and digits (0-9). The ·Originator· MUST not exceed 255 characters in length.

The SiLA 2 Specification establishes naming conventions for the ·Originator· to avoid the possibility of two published ·Features· with the same ·Feature Identifier· having the same ·Originator·. The naming conventions below describe how to create unique ·Originator· names.

In general, an ·Originator· item begins with the top level domain name of the organization and then the organization's domain and then any subdomains, listed in reverse order and separated by dots

(“.”). Subsequent components of the ·Originator· item vary according to an organization's own internal naming conventions.

In some cases, the Internet domain name may not result in a valid ·Originator· item. Here are some suggested conventions for dealing with these situations:

- If the domain name contains a hyphen, or any other special character not allowed in an ·Originator·, remove it.
- If any of the resulting ·Originator· item components start with a digit, or any other character that is not allowed as an initial character, have it removed.

The ·Originator· is not meant to imply where the ·Feature· is stored within the Internet. The suggested convention for generating unique ·Originators· is merely a way to piggyback a ·Originator· naming convention on top of an existing, widely known unique name registry instead of having to create a separate registry for ·Originators·.

Examples

[COMPLETE; as of 0.1]

- org.silastandard
- de.tuberlin.bioprocess
- com.anyvendor

Category

[COMPLETE; as of 0.1]

[Definition: Category] The **Category** is mandatory, but can be set to “none”. It MAY be used to group ·Features· and assign them with a logical or semantic category or category and sub-categorie(s). The main purpose for **Categories** is to group ·Features· into domains of application.

Item	Description
Category	The ·Category· MUST consist of one word (a simple category) or multiple words (category and sub-category / categories) separated by dots (“.”). Each word MUST start with a lower-case letter (a-z), followed by any number of lower-case letters (a-z) and digits (0-9). The ·Category· MUST not exceed 255 characters in length.

Currently only two·Categories· are standardized by SiLA 2. Additional ·Categories· might be added in the future.

Category	Description
none	The “none” ·Category· MUST be used for all ·Features· that do not belong to a specific ·Category·.
core	The “core” ·Category· MUST be used for all ·Features· that are independent of the type of ·SiLA Server· and in principle can be implemented by any ·SiLA Server·.

Other ·Categories· can be defined arbitrarily, see also examples below.

Examples

[COMPLETE; as of 0.1]

Examples of categories:

Category	Description
analysis	Features that deal with behaviors required for analytical measurement methods and data analysis.
analysis.hplc	Features that deal with behaviors required for analytical measurement methods and data analysis used for HPLC.
analysis.ph	Features that deal with behaviors required for analytical measurement methods and data analysis used for pH measurements.
synthesis	Features that deal with behaviors of e.g. PCR cyclers.
weighing	Features that deal with behaviors of e.g. Balances.
dispensing	Features that deal with behaviors of e.g. Dispensers, pipettors, washers, syringe pumps, acoustic dispensers, … .
heatingcooling	Features that deal with behaviors of e.g. Heating/cooling Units.
pressurizing	Features that deal with behaviors of e.g. Vacuum Pumps or Evaporators.
mixing	Features that deal with behaviors of e.g. Shakers or Centrifuges.
samplehandling	Features that deal with behaviors of e.g. Conveyor Belts, Robots, Lid Handlers, Sealers, Peelers, Piercers, Cappers / Decappers, Tube Pichers / Punchers.
storing	Features that deal with behaviors of e.g. Stackers, Hotel or Incubators.
imaging	Features that deal with behaviors of e.g. Readers / Imagers, Code Reader, Cameras.
printing	Features that deal with behaviors of e.g. Code Labelers.

Feature Definition Language

[COMPLETE; as of 0.1]

As described earlier in this specification, ·Features· MUST be human and machine readable. In order to store and process SiLA ·Features· in both a programmatic and human readable manner, SiLA decided to use a text based XML format to persist ·Features·. In order to be able to validate this XML based ·Feature Definition·, SiLA specifies an XML schema for ·Feature Definitions·, aka the ·Feature Definition Language·.

[Definition: Feature Definition Language] The **Feature Definition Language** is a way to store ·Feature Definitions· programmatically in an XML (text) based, human and machine readable way. The XML schema can be found on GitLab as FeatureDefinition.xsd.

With regards to a ·SiLA Data Type·, a ·Feature Definition· SHALL be of a ·SiLA String Type· with a "Schema"-·Constraint· of type "Xml" and an URL referring to the FeatureDefinition.xsd.

The SiLA Service Feature

[COMPLETE; as of 0.1]

[Definition: SiLA Service Feature] The **SiLA Service Feature** is the ·Feature· each ·SiLA Server· MUST implement. Each ·SiLA Server· MUST at least implement the **SiLA Service Feature** with ·Major Feature Version· equals one (1). It is the entry point to a ·SiLA Server· and helps to discover the ·Features· it implements. The **SiLA Service Feature** specifies ·Commands· and ·Properties· to discover the ·Features· a ·SiLA Server· implements as well as details about the ·SiLA Server·.

The normative ·Feature Definition· of the ·SiLA Service Feature· ("SiLA Service") can be found on GitLab as SiLAService-v1_0.sila.xml.

The Connection Configuration Service Feature

[COMPLETE; as of 1.1]

[Definition: Connection Configuration Service Feature] The **Connection Configuration Service Feature** is a ·Feature· that all ·SiLA Servers· conforming to ·SiLA 2 Version· equal to or greater than "1.1" SHALL implement. The **Connection Configuration Service Feature** specifies ·Commands· and ·Properties· to configure the ·Connection Method· of a ·SiLA Server·.

The normative ·Feature Definition· of the ·Connection Configuration Service Feature· ("Connection Configuration Service") can be found on GitLab as ConnectionConfigurationService-v1_0.sila.xml.

Error Categories

[COMPLETE; as of 0.2]

There are four types of errors that can happen when a ·SiLA Client· communicates with a ·SiLA Server· over a ·Connection·:

- ·Validation Error·
- ·Execution Error·
- ·Framework Error·
- ·Connection Error·

All types of errors, except for the ·Connection Error· are always issued by the ·SiLA Server·, but never by the ·SiLA Client·.

The ·Connection Error· is different, as it is not actively issued by the ·SiLA Server· nor the ·SiLA Client·, but by the underlying infrastructure (such as the communication network).

Validation Error

[COMPLETE; as of 0.1]

[Definition: Validation Error] A **Validation Error** is an error that occurs during the validation of ·Parameters· before executing a ·Command·.

Before executing a ·Command·, a ·SiLA Server· MUST validate all ·Parameters· and MUST issue a ·Validation Error· in case of invalid or missing ·Parameters·. The ·Validation Error· MUST include the ·Fully Qualified Command Parameter Identifier·, human readable information in the American English language (see Internationalization) why the ·Parameter· was invalid and SHOULD provide proposals for how to resolve the error (e.g. present a valid ·Parameter· range to the user).

Execution Error

[COMPLETE; as of 0.2]

[Definition: Execution Error] An **Execution Error** is an error which occurs during a ·Command· execution, a ·Property· access or an error that is related to the use of ·SiLA Client Metadata·.

Defined Execution Error

[COMPLETE; as of 0.2]

[Definition: Defined Execution Error] A **Defined Execution Error** is an ·Execution Error· that has been defined by the ·Feature Designer· as part of the ·Feature·. **Defined Execution Errors** enable the ·SiLA Client· to react more specifically to an ·Execution Error·, as the nature of the error as well as possible recovery procedures are known in better detail.

·Defined Execution Errors· enable the ·Feature Designer· to design error situations and allow the ·SiLA Client· to implement situation specific and more explicit error handling routines.

The ·Defined Execution Error· MUST include its ·Fully Qualified Defined Execution Error Identifier·, human readable information in the American English language (see Internationalization) about the error and SHOULD provide proposals for how to resolve the error.

Examples for Defined Execution Errors in the Command Execution Context

[COMPLETE; as of 0.2]

Plate Handling Feature

Let us assume we have a ·Feature· design that designs a plate handling behavior. Note that a good ·Feature· design is independent of an actual implementation. Depending on the actual implementation of the plate handling behavior in the ·SiLA Server·, this could be implemented by different types of a robot, a conveyor belt, or any other “logistics” system, yet unknown at the time of the ·Feature· design.

The design specifies a ·Command· to move a plate from position A to position B. The requirement to execute this ·Command· is that the handler is not busy, there is a plate at position A and position B is not occupied. All these three preconditions can be foreseen, can be designed into the ·Feature· and therefore designed as ·Defined Execution Error· by the ·Feature Designer·:

- Robot is busy with another move

- No source plate available (at position A)
- No space available (at position B)

However, while moving the plate from A to B, the handler could run into an unforeseen error situation. Such an error would be returned as ·Undefined Execution Error·, as the specific details and nature of the error could not be foreseen by the ·Feature· design.

Authorization Service Feature

Let us assume we have a ·SiLA Server· that implements the AuthorizationService ·Feature· and another ·Feature· with a ·Command· “X”. The AuthorizationService specifies a “UnauthorizedAccess” ·Defined Execution Error·. If an unauthorized ·SiLA Client· tries to execute the ·Command· “X”, an “UnauthorizedAccess” ·Defined Execution Error· will be issued, containing a reference to the AuthorizationService ·Feature· as the origin of the error.

Examples for Defined Execution Errors in the Property Access Context

[COMPLETE; as of 0.2]

Sensor Off

The temperature sensing ·Feature· that has the ·Observable Property· “Temperature” defines the “Sensor Off” ·Defined Execution Error· that describes that the sensor has to be turned on first.

Examples for Defined Execution Errors in the SiLA Client Metadata Context

[COMPLETE; as of 0.2]

Invalid Lock Identifier

A Lock Controller ·Feature· defines a ·SiLA Client Metadata· “Lock Identifier”, which has to be sent with every (locked) call. It also defines a ·Defined Execution Error· “Invalid Lock Identifier” that is issued when an according ·SiLA Client Metadata· with the correct data type has been sent but the contained Lock ·Identifier· is invalid.

Undefined Execution Error

[COMPLETE; as of 0.2]

[Definition: Undefined Execution Error] Any ·Execution Error· which is not a ·Defined Execution Error· is an **Undefined Execution Error**.

These types of errors are implementation dependent and occur unexpectedly and cannot be foreseen by the ·Feature Designer· and hence cannot be not specified as part of the ·Feature·.

The ·Undefined Execution Error· MUST include human readable information in the American English language (see Internationalization) about the error and SHOULD provide proposals for how to resolve the error.

Framework Error

[COMPLETE; as of 0.2]

[Definition: Framework Error] A **Framework Error** is an error which occurs when a ·SiLA Client· accesses a ·SiLA Server· in a way that violates the SiLA 2 specification. The **Framework Error**

MUST include human readable information in the American English language (see Internationalization) about the error and SHOULD provide proposals for how to resolve the error.

The following ·Framework Errors· can occur, as listed in the following chapters:

Command Execution Not Accepted

[COMPLETE; as of 0.1]

[Definition: Command Execution Not Accepted Error] The **Command Execution Not Accepted Error** is a ·Framework Error· and MUST be issued in case the ·SiLA Server· does not allow the ·Command· execution.

Invalid Command Execution UUID

[COMPLETE; as of 0.1]

[Definition: Invalid Command Execution UUID Error] The **Invalid Command Execution UUID Error** is a ·Framework Error· which MUST be issued when a ·SiLA Client· is trying to get or subscribe to ·Command Execution Info·, ·Intermediate Command Response· or ·Command Response· of an ·Observable Command· with an Invalid ·Command Execution UUID·.

Command Execution Not Finished

[COMPLETE; as of 0.1]

[Definition: Command Execution Not Finished Error] The **Command Execution Not Finished Error** is a ·Framework Error· and MUST be issued when a ·SiLA Client· is trying to get the ·Command Response· of an ·Observable Command· when the ·Command· has not been finished yet.

Invalid Metadata

[COMPLETE; as of 0.2]

[Definition: Invalid Metadata Error] The **Invalid Metadata Error** is a ·Framework Error· and MUST be issued by a ·SiLA Server· if a required ·SiLA Client Metadata· has not been sent to the ·SiLA Server· or if the sent ·SiLA Client Metadata· has the wrong ·SiLA Data Type· (e.g. not the one that was specified in the ·Feature Definition· for the respective ·SiLA Client Metadata·).

No Metadata Allowed

[COMPLETE; as of 0.2]

[Definition: No Metadata Allowed Error] The **No Metadata Allowed Error** is a ·Framework Error· and MUST be issued when a ·SiLA Server· receives a call of the ·SiLA Service Feature· that contains ·SiLA Client Metadata·.

Connection Error

[COMPLETE; as of 0.1, updates: 1.1]

[Definition: Connection Error] SiLA 2 treats any error with the ·Connection· between a ·SiLA Client· and a ·SiLA Server· as a **Connection Error**. In contrast to the other error types, **Connection**

Errors are not issued by the ·SiLA Server· nor the ·SiLA Client·, but by the underlying infrastructure (such as the communication network, the operating system, etc.).

If a ·Connection Error· happens, the ·SiLA Server· and ·SiLA Client· MUST continue to function normally in any case. Both ·SiLA Server· and ·SiLA Client· MUST handle a ·Connection Error· gracefully, at anytime during the ·Lifetime of a SiLA Server·.

In case of a ·Connection Error·, the ·SiLA Server· MUST stop sending ·Observable Property· changes or ·Command Execution Info· events from ·Observable Command· executions to the affected ·SiLA Client·. The ·SiLA Server· MUST, however, continue processing ·Observable Commands· that the ·SiLA Client· has initiated and that were already accepted for ·Command· execution by the ·SiLA Server·. All ·Command Execution UUIDs· MUST remain valid as specified, so that a ·SiLA Client· is still able to use it after having re-connected to the ·SiLA Server·.

The consequences of a ·Connection Error· happening in the following situations are:

- While reading a ·Property·: A ·SiLA Client· would have to re-read the ·Property·.
- While subscribed to an ·Observable Property·: ·SiLA Server· MUST stop sending new ·Property· values. A ·SiLA Client· would have to re-subscribe to continue to get events in the future.
- ·Unobservable Command·: In case all ·Parameters· were received by the ·SiLA Server·, the ·Command· MUST be executed, but the ·Command Response· will be lost. If not all ·Parameters· have been properly submitted as the ·Connection Error· happened while sending the ·SiLA Client Request·, the ·SiLA Server· MUST NOT start to execute the ·Command·.
- ·Observable Command·: The ·SiLA Server· MUST only start the ·Command· execution once it is sure the ·SiLA Client· has properly received the ·Command Execution UUID·. A ·SiLA Client· can be sure that the ·Command· executes in case there was no ·Connection· or no ·Validation Error· until it got the ·Command Execution UUID· from the ·SiLA Server·. The ·SiLA Server· MUST stop sending ·Command Execution Info· events and ·Intermediate Command Responses· in case a ·SiLA Client· was subscribed to them. It is the responsibility of the ·SiLA Client· to re-subscribe.

The ·SiLA Client· has to re-establish the ·Connection· and re-subscribe to ·Observable Properties· or to the ·Command Execution Info· or ·Intermediate Command Responses· of ·Observable Commands·, in case the ·SiLA Client· likes to continue receiving data from the ·SiLA Server·.

SiLA Server Discovery

[COMPLETE; as of 0.1]

[Definition: SiLA Server Discovery] **SiLA Server Discovery** is a set of mechanisms that can be used by a ·SiLA Server· to advertise itself (i.e. its ·Address·) in the communication network to a ·SiLA Client·. Using the **SiLA Server Discovery** mechanism a ·SiLA Client· is able to discover the ·Address· of a ·SiLA Server· in the communication network. The goal of ·SiLA Server Discovery· is to enable small, ad-hoc automation setups in labs.

·SiLA Server Discovery· is an important part of this specification that will increase the accessibility of SiLA to common users (that are usually not IT experts), especially in ad-hoc laboratory automation setups.

·SiLA Server Discovery· MUST be implemented by each ·SiLA Server·. However, a ·SiLA Server· MAY provide means to disable ·SiLA Server Discovery·. It is RECOMMENDED that all ·SiLA Servers· have ·SiLA Server Discovery· enabled by default, but all ·SiLA Devices· MUST have ·SiLA Server Discovery· enabled by default.

·SiLA Server Discovery· allows a ·SiLA Client· to discover the presence of a ·SiLA Server·. ·SiLA Server Discovery· MUST be designed to work in local network setups.

·SiLA Server Discovery· in global network setups will require additional systems, such as service registries, that are not part of this specification document.

Feature Discovery

[COMPLETE; as of 0.1]

[Definition: SiLA Feature Discovery] **SiLA Feature Discovery** allows a ·SiLA Client· to discover the ·Features· of a ·SiLA Server·. A ·SiLA Server· MUST enable any ·SiLA Client· to discover available ·Features· of a ·SiLA Server· through the ·SiLA Service Feature·, therefore a ·SiLA Server· MUST always implement the ·SiLA Service Feature·.

Internationalization

[COMPLETE; as of 0.1]

Any part of a ·Feature Definition·, that is supposed to be human readable, such as ·Feature Identifier·, ·Display Names·, ·Descriptions·, ·Commands·, ·Command Parameters· and ·Properties· MUST be in the American English language, as specified by language tag “en-US” (English as used in the United States) according to IETF BCP 47 language tag.

Any value of a ·Parameter·, ·Property· (including its physical unit) or error message that is supposed to be human readable text by design of the ·Feature·, MUST be provided by the ·SiLA Server· or ·SiLA Client· implementation in the American English language, as specified by tag “en-US” (English as used in the United States) according to IETF BCP 47 language tag.

Any ·SiLA Server· MAY support additional languages and indicate this to the ·SiLA Client· by implementing the “InternationalizationService” ·Feature·. It is RECOMMENDED that a ·SiLA Server· implements this ·Feature·.

Security

[COMPLETE; as of 0.1]

SiLA 2 specifies some ·Features· in order to increase data integrity, especially for operation in highly regulated areas such as those falling under GLP or GMP or other regulations.

Encryption

[COMPLETE; as of 0.1]

Communication between ·SiLA Client· and ·SiLA Server· MUST be secured by encryption.

Authentication

[COMPLETE; as of 0.1]

In SiLA 2, authentication is the process of actually confirming the identity of a ·SiLA Client·, executed by a ·SiLA Server·.

It is RECOMMENDED that a ·SiLA Server· always authenticates the identity of a ·SiLA Client· by implementing the ·Feature· “AuthenticationService”.

Authorization

[COMPLETE; as of 0.1]

In SiLA 2, authorization is the function of access control, executed by a ·SiLA Server· in order to grant or deny access to the ·SiLA Server· by a ·SiLA Client·.

It is RECOMMENDED that a ·SiLA Server· always authorizes the access of a ·SiLA Client· by implementing the ·Feature· “AuthorizationService”.

Audit Trail

[COMPLETE; as of 0.1]

To improve data integrity and in order to fulfill regulatory requirements, SiLA specifies means to implement an audit trail function.

A ·SiLA Server· MAY support an audit trail function by implementing the ·Feature· “AuditTrailService”.

Versioning Strategy

[COMPLETE; as of 0.1]

One of the most important goals for SiLA 2 is the full backwards compatibility and - based on a best effort - full forwards compatibility.

The term “backwards compatibility” in this context means that a ·SiLA Server· based on an older ·SiLA 2 Version· MUST be usable by a ·SiLA Client· based on a newer ·SiLA 2 Version·. By contrast, the term “forwards compatibility” in this context means that a ·SiLA Server· based on a newer ·SiLA 2 Version· MUST be usable by a ·SiLA Client· based on an older ·SiLA 2 Version·.

Both the SiLA 2 Specification and complying ·Features· will be versioned separately. ·Feature· versioning is documented thoroughly in the chapter Feature Version.

The core specification, Part (A) - Overview, Concepts and Core Specification (this document), MUST be maintained in a way to enable full backwards and forwards compatibility.

The Part (B) - Mapping Specification MUST specify mechanisms to ensure full backwards compatibility.

The Part (B) - Mapping Specification MUST specify mechanisms to ensure full forwards compatibility, or - if not possible - mechanisms to “fail fast and safe”. The term “fail fast and safe” in

this context means that in case of incompatibility between ·SiLA 2 Versions· or ·Feature Versions· of the ·SiLA Client· and ·SiLA Server·, there will be a deterministic behavior.

Fully Qualified Identifier

[COMPLETE; as of 0.2]

[Definition: Fully Qualified Identifier] Each ·Feature· and its components (·Commands·, ·Properties·, ·SiLA Data Types·, etc) SHALL be identifiable by a **Fully Qualified Identifier**. A **Fully Qualified Identifier** is guaranteed to be a universally unique identifier, see also Uniqueness of Fully Qualified Identifiers. A Fully **Qualified Identifier** MUST be a string of UNICODE characters up to a maximum of 2048 characters in length.

Uniqueness of Fully Qualified Identifiers

[COMPLETE; as of 1.0]

In general, ·Fully Qualified Identifiers· have to be universally unique. Even though ·Fully Qualified Identifiers· contain both lower (a-z) and upper (A-Z) case letters, the uniqueness MUST always be checked without taking lower and upper case into account (i.e. apply “a” = “A”, “b” = “B”, … “z” = “Z” when comparing ·Fully Qualified Identifiers·).

Fully Qualified Feature Identifier

[COMPLETE; as of 0.2]

[Definition: Fully Qualified Feature Identifier] A **Fully Qualified Feature Identifier** is a name that uniquely identifies a ·Feature· among all potentially existing ·Features·. It is a combination of the ·Originator·, ·Category·, ·Feature Version· and ·Feature Identifier· in the form of
Fully Qualified Feature Identifier != ·Originator· + “/” + ·Category·+ “/” + ·Feature Identifier· + “/” + “v” + ·Major Feature Version·.

Example: **org.silastandard/core/SiLAService/v1**

Fully Qualified Command Identifier

[COMPLETE; as of 0.2]

[Definition: Fully Qualified Command Identifier] A **Fully Qualified Command Identifier** is a name that uniquely identifies a ·Command· among all potentially existing ·Commands·.
Fully Qualified Command Identifier != ·Fully Qualified Feature Identifier· + “/” + ”Command” + “/” ·Command Identifier·.

Example: **org.silastandard/core/SiLAService/v1/Command/GetFeatureDefinition**

Fully Qualified Command Parameter Identifier

[COMPLETE; as of 0.2]

[Definition: Fully Qualified Command Parameter Identifier] A **Fully Qualified Command Parameter Identifier** is a name that uniquely identifies a ·Command Parameter· among all

potentially existing ·Command Parameters·.
Fully Qualified Command Parameter Identifier != ·Fully Qualified Command Identifier· + “/” + ”Parameter” + “/” ·Command Parameter Identifier·.

Fully Qualified Command Response Identifier

[COMPLETE; as of 0.2]

[Definition: Fully Qualified Command Response Identifier] A **Fully Qualified Command Response Identifier** is a name that uniquely identifies a ·Command Response· among all potentially existing ·Command Responses·.
Fully Qualified Command Response Identifier != ·Fully Qualified Command Identifier· + “/” + ”Response” + “/” ·Command Response Identifier·.

Fully Qualified Intermediate Command Response Identifier

[COMPLETE; as of 0.2]

[Definition: Fully Qualified Intermediate Command Response Identifier] A **Fully Qualified Intermediate Command Response Identifier** is a name that uniquely identifies a ·Intermediate Command Response· among all potentially existing ·Intermediate Command Responses·.
Fully Qualified Intermediate Command Response Identifier != ·Fully Qualified Command Identifier· + “/” + ”IntermediateResponse” + “/” ·Intermediate Command Response Identifier·.

Fully Qualified Defined Execution Error Identifier

[COMPLETE; as of 0.2]

[Definition: Fully Qualified Defined Execution Error Identifier] A **Fully Qualified Defined Execution Error Identifier** is a name that uniquely identifies a ·Defined Execution Error· among all potentially existing ·Defined Execution Errors·.
Fully Qualified Defined Execution Error Identifier != ·Fully Qualified Feature Identifier· + “/” + ”DefinedExecutionError” + “/” ·Defined Execution Error Identifier·.

Example:
org.silastandard/core/InternationalizationService/v1/DefinedExecutionError/UnsupportedLanguage

Fully Qualified Property Identifier

[COMPLETE; as of 0.2]

[Definition: Fully Qualified Property Identifier] A **Fully Qualified Property Identifier** is a name that uniquely identifies a ·Property· among all potentially existing ·Properties·.
Fully Qualified Property Identifier != ·Fully Qualified Feature Identifier· + “/” + ”Property” + “/” ·Property Identifier·.

Fully Qualified Custom Data Type Identifier

[COMPLETE; as of 0.2]

[Definition: Fully Qualified Custom Data Type Identifier] A **Fully Qualified Custom Data Type Identifier** is a name that uniquely identifies a ·Custom Data Type· among all potentially existing ·Custom Data Types·.
Fully Qualified Custom Data Type Identifier != ·Fully Qualified Feature Identifier· + “/” + ”DataType” + “/” ·Custom Data Type Identifier·.

Fully Qualified Metadata Identifier

[COMPLETE; as of 0.2]

[Definition: Fully Qualified Metadata Identifier] A **Fully Qualified Metadata Identifier** is a name that uniquely identifies a ·SiLA Client Metadata· among all potentially existing ·SiLA Client Metadata·.
Fully Qualified Metadata Identifier != ·Fully Qualified Feature Identifier· + “/” + ”Metadata” + “/” ·Metadata Identifier·.
Example: **org.silastandard/core/AuthorizationService/v1/Metadata/AccessToken**

Recurrent Terminology

[COMPLETE; as of 0.1]

In SiLA 2, there are certain recurrent terms that are used at different places throughout the specification documents. For better readability, they are explained here once:

Identifier

[COMPLETE; as of 0.1]

[Definition: Identifier] An **Identifier** is a name that serves as explicit identifier for different components in SiLA 2. For example, each ·Feature· and its components (e.g. ·Commands·, ·Command Parameters·, etc.) MUST be identifiable by an **Identifier**. An **Identifier** MUST be a string of UNICODE characters, start with an upper-case letter (A-Z) and MAY be continued by lower and upper-case letters (A-Z and a-z) and digits (0-9) up to a maximum of 255 characters in length.

Uniqueness of Identifiers

[COMPLETE; as of 1.0]

In general, ·Identifiers· have to be unique within a scope. For the different types of ·Identifiers·, the scope is defined individually, at the respective definition of that ·Identifier· type. Even tough ·Identifiers· contain both lower (a-z) and upper (A-Z) case letters, the uniqueness within a certain scope MUST always be checked without taking lower and upper case into account (i.e. apply “a” = “A”, “b” = “B”, … “z” = “Z” when comparing ·Identifiers·).

Display Name

[COMPLETE; as of 0.1]

[Definition: Display Name] Each ·Feature· and many of its components (e.g. ·Commands·, ·Command Parameters·, etc.) MUST have a human readable **Display Name**. This is the name that will be visible to the user. A **Display Name** MUST be a string of UNICODE characters of maximum 255 characters in length. The **Display Name** MUST be human readable text in American English (see also Internationalization).

Description

[COMPLETE; as of 0.1]

[Definition: Description] A **Description** is a human readable text that describes the behavior and provides additional information about the described element (e.g. ·Feature·, ·Command·, etc.) with details. A **Description** MUST be a string of UNICODE characters of any number of characters. The **Description** MUST be human readable text in American English (see also Internationalization).

Best Practice

[COMPLETE; as of 0.1]

As a best practice, a ·Description· SHOULD consist of complete sentences that are easy to read. Special care should be taken to make it easy to understand as well. The ·Description· SHOULD also contain proper interpunction (commas, dots).

Parameter

[COMPLETE; as of 0.1]

[Definition: Parameter] **Parameters** are used to parameterize specific SiLA components (e.g. ·Commands·). Each **Parameter** MUST have a ·SiLA Data Type· assigned.

UUID

[COMPLETE; as of 1.0]

[Definition: UUID] A **UUID** is a **U**niversally **U**nique **ID**entifier according to RFC 4122. SiLA always uses the **UUID** in its string representation (e.g. “f81d4fae-7dec-11d0-a765-00a0c91e6bf6”), as specified by the formal definition of the **UUID** string representation in RFC 4122. It is RECOMMENDED to always use lower case letters (a-f). In any case, comparisons of **UUID**s in their string representation must always be performed ignoring lower and upper case, i.e. “a” = “A”, “b” = “B”, … , “f” = “F”.

Normative References

[COMPLETE; as of 0.1]

01. Creative Commons Attribution-ShareAlike (CC BY-SA 4.0)
02. [RFC2119] Key words for use in RFCs to Indicate Requirement Levels
03. http://www.unicode.org/

04. https://en.wikipedia.org/wiki/Number
05. https://en.wikipedia.org/wiki/Double-precision_floating-point_format
06. https://en.wikipedia.org/wiki/Boolean_data_type
07. https://en.wikipedia.org/wiki/Binary_file
08. https://en.wikipedia.org/wiki/ISO_8601
09. https://en.wikipedia.org/wiki/List_of_UTC_time_offsets
10. https://en.wikipedia.org/wiki/Coordinated_Universal_Time
11. [RFC2045] Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies
12. [RFC 4122] A Universally Unique IDentifier (UUID) URN Namespace.
13. AnIML - Analytical Information Markup Language (AnIML) is the emerging ASTM XML standard for analytical chemistry data. https://www.animl.org/
14. XML - https://www.w3.org/TR/xml/
15. XML schema - https://www.w3.org/XML/Schema

= = = END OF NORMATIVE PART = = =